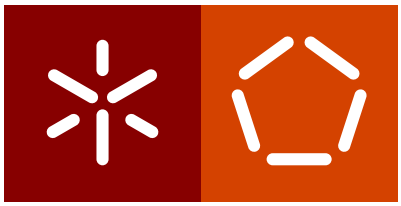
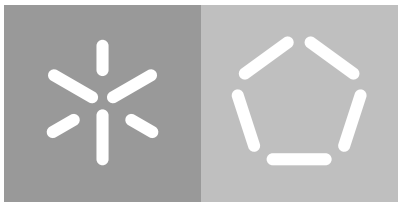


Universidade do Minho
Escola de Engenharia
Departamento de Informática

Marcelo José Rodrigues Gonçalves

Model-based Testing of User Interfaces

October 2017



Universidade do Minho
Escola de Engenharia
Departamento de Informática

Marcelo José Rodrigues Gonçalves

Model-based Testing of User Interfaces

Master dissertation
Master Degree in Computer Science

Dissertation supervised by
José Creissac Campos
Camille Fayollas

October 2017

This dissertation was supported by Project ”NORTE-01-0145-FEDER-000016”, financed by the North Portugal Regional Operational Programme (NORTE 2020), under the PORTUGAL 2020 Partnership Agreement, and through the European Regional Development Fund (ERDF).

Acknowledgements

I would like to thank my advisers, Professor José Creissac Campos and Camille Fayollas, for the opportunity, support and suggestions given during the development of this dissertation. Thanks for the support provided by the Research Grant level B1-D, with reference 6049/BI_B1D/17.

Special thanks to my parents, my brothers and my friends for all the encouragement and unconditional support.

Finally, to all those who were present during my academic formation, thank you.

Abstract

The *Graphical User Interface (GUI)* is crucial for the success of an interactive system. Incorrect operation of the GUI may inhibit the proper functioning of the system. Ensuring the quality of a system is essential to its success. A practical way to ensure it is through software testing. *Model-Based Testing (MBT)* is a black-box technique that compares the behaviour of the system under test against the behaviour of an oracle (the system's model). These tests are intended to verify that the implementation of a software meets its specification.

Applying MBT tools to GUIs enables the systematic testing of the system by automatically simulating user actions on the user interface. However, applying the MBT process to GUIs creates several challenges such as the mapping between the actions in the model and the actions in the application interface. This dissertation will focus on the model-based testing of graphical user interfaces. The main objective is to further the development of the TOM Framework. The TOM Framework supports the process of MBT applied to GUIs, in particular, web-based GUIs, enabling the creation and execution of user interfaces tests and thus increasing the probability of error detection in the tested interfaces.

Resumo

A interface gráfica do utilizador é imprescindível para o sucesso de um sistema interativo. O incorreto funcionamento da interface gráfica pode inibir o bom funcionamento da aplicação e, consequentemente, do sistema de software. Assegurar a qualidade de um sistema é essencial para o seu sucesso. Uma maneira prática de o garantir é através dos testes de software. Os testes baseados em modelos são uma técnica de caixa-preta que compara o comportamento do sistema sob teste com o comportamento de um oráculo (o modelo do sistema). Estes testes destinam-se a verificar se a implementação do software cumpre as suas especificações.

A aplicação de ferramentas que suportem testes baseados em modelos em interfaces gráficas do utilizador permite o teste sistemático do sistema, simulando automaticamente ações realizadas pelo utilizador na interface disponibilizada. No entanto, a aplicação do processo dos testes baseados em modelos em interfaces gráficas cria vários desafios, como por exemplo, o mapeamento entre as ações no modelo e as ações na interface da aplicação. Esta dissertação incidirá sobre os testes baseados em modelos e a sua aplicação a interfaces gráficas do utilizador. O principal objectivo é continuar o desenvolvimento da Framework TOM. A Framework TOM suporta a aplicação do processo de testes baseados em modelos a interfaces gráficas do utilizador, permitindo a criação e execução de testes em interfaces e aumentando a probabilidade de detecção de erros nas interfaces testadas.

Contents

List of Figures

List of Tables

List of Listings

Acronyms

A

API Application Program Interfaces.

C

CLI Command Line Interface.

CORS Cross-origin resource sharing.

D

DLS Domain Specific Language.

F

FCU Flight Control Unit.

FCUS Flight Control Unit Software.

FSM Final State Machines.

G

GUI Graphical User Interface.

H

HTML Hypertext Markup Language.

I

IRIT Institut de Recherche en Informatique de Toulouse.

ISTQB International Software Testing Qualification Board.

J

JSON JavaScript Object Notation.

JVM Java Virtual Machine.

M

MBT Model-Based Testing.

P

PBGT Pattern Based GUI Testing.

PTS Presentation Task Set.

R

REST Representational State Transfer.

S

SCXML State Chart XML.

SUT System Under Test.

Chapter 1

Introduction

Software applications play a crucial role in modern society. Daily, at work or during leisure time, we interact with computers. Such systems allow us to be more efficient, but also enable us to spend pleasure moments, for example playing or watching movies. It is thus essential that software-based systems do not present any points of failure, as such failures can have irremediable costs for users. Ensuring the quality of their systems is a growing concern of developers. In the case of interactive systems, analysing the *Graphical User Interface (GUI)* is particularly relevant, as it constitutes a critical part of the system: the medium through which the system's features are made available to users. As stated by Memon (2001) GUIs have become nearly ubiquitous as a means of interacting with software systems. The GUI, also called front-end, allows the user to perform sequences of events (mouse movements, menu selections, etc) (Yang, 2011). In response to these events, the GUI replies to the user or interacts with the underlying code through messages or method calls to obtain the answer to the user's actions.

An incorrect functioning of the GUI can derail the smooth operation of the application and consequently of the software system, which can lead to large losses. So, to ensure the quality of their systems, developers are increasingly investing resources in software testing tools to test their applications' GUIs.

As stated by the IEEE Software Engineering Body of Knowledge (ISO94):

> *"Testing is an activity performed for evaluating product quality, and for improving it, by identifying defects and problems."*

That is, the purpose of testing is evaluating if a developed product obeys its specifications and works as intended. So software testing, particularly GUI testing, is crucial to ensuring the quality of the software. However, due to the increasing complexity of applications, the testing process tends to be very time-consuming and expensive. Being a task that commonly appears later in the project, when errors occur their correction is more difficult (Blackburn et al., 2004), which again leads to expensive costs. Hence, it is frequent that the applications are available to the users without being properly tested, allowing the presence of errors in the final product.

It is important to provide tools that support automatic analysis of applications, including their user interface layer (Campos et al., 2016). These tools cannot replace the intelligence of software testers, as it will always be up to them the decision whether a concrete test failure is an error or not. However, without these tools, analysing complex applications at a reasonable cost will never be possible (Blackburn et al., 2004).

Software tests can be performed to validate a design (in the specific case of user interfaces, usually via user testing) or to check that a concrete implementation is correct. A particular case of the latter is model-based testing (Blackburn et al., 2004). Model-based testing allows automating the testing process by comparing the implemented system with a model of itself (an "oracle").

The model gives us a description of the correct behaviour of the application, so we use this to determine where incorrect behaviour occurs in the implemented system, by looking for situations which violate the model (Bowen and Reeves, 2013). To do this, test cases (sequences of actions) are generated from the "oracle" and performed on the application. Then, the behaviour of the application during the test case execution is compared with the oracle's prediction. With this kind of test cases, the task of testing can be more automated, which helps reduce both time and effort required.

Many tools have been proposed to perform Model-based Testing of user interfaces (Nguyen et al., 2014; Moreira and Paiva, 2014). One example is the TOM Framework, being developed at the University of Minho. This tool has a modular architecture, so, each of the modules (features) can operate independently, for an easier maintenance of the individual modules (Pinto, 2017). This modular approach also means that the tool can be adapted to different contexts. The tool's main target is the generation of test cases using a model (TOM Generator). However, the tool also provides a component to help the creation of the model of the system (TOM Editor). Although the central point of the tool is the testing of Web Applications (Rodrigues, 2015), some of its modules have also been adjusted to integrate with task modelling tools developed at the *Institut de Recherche en Informatique de Toulouse (IRIT)* (Campos et al., 2016).

1.1 Motivation

Testing graphical user interfaces is increasingly difficult because of their increasing complexity. Model-based tests allow a high level of test systematization as the test cases generation, test cases execution and the comparison of the oracle's prediction with the real results can all be automated.

There are several challenges in the application of model-based tests to graphical user interfaces. For example, the mapping between the events in the model and the concrete actions in the interface is necessary. Hence, it is important to develop an environment for the generation and execution

of test cases, adaptable and easy to use, to create more reliable and higher quality GUIs for the end user. The developed environment should automate the process of testing applied to graphical interfaces covering the whole process of MBT, from the creation of the model to the production of reports of the generated test cases, once executed.

1.2 Main Goals

The goal of this dissertation is to continue the development of the TOM Framework. The main objectives to be achieved are:

- To optimise the test case generation algorithms, both for positive tests and for negative tests (tests that simulate user errors). As case study, the Electronic Flight Information System Control Panel, a cockpit component of the Airbus A380 (or equivalent system) will be used;
- Improve the architecture of the TOM Framework, by developing a Web Services layer supporting access to its functionalities, and deepen the integration of the TOM Framework with the tools under development in IRIT. Currently, the tool takes input data through files, it is intended that the tool accepts input through requests to a server, thus allowing greater integration between the developed tool and other applications that want to use its facilities;
- Considering the Web Services layer to be developed, build a Web Application (TOM App) to support the management of the models, accepting test generation requests, receiving the generation results and obtaining the generated tests;

1.3 Document Organization

The remaining document is structured as follows:

- Chapter 1 - Introduction - Contains a description and the objectives of the project.
- Chapter 2 - Software Testing - Provides an introduction to the topic of software testing to contextualize the project.
- Chapter 3 - TOM Generator - Presents the features of the TOM Generator component.
- Chapter 4 - Contributions - Presents all the contributions made throughout this dissertation.
- Chapter 5 - Case Studies - Presents the use of the Framework in several case studies.
- Chapter 6 - Conclusions - Contains an analysis of the developed work and suggestions for future work.

Chapter 2

Software Testing

The software development cycle involves ambiguity, assumptions, and miscommunication between humans. Besides, each change made to a piece of software, each new features, each effort to correct a defect, raises the possibility of error. Every defect in the software increases the risk that the software does not match the requirements. Quality is an increasingly relevant factor in software development, making the task of testing one of the most essential in the development cycle. However, the high complexity of the systems, as well as tight development deadlines, make this task increasingly challenging (Blackburn et al., 2004).

Software testing is crucial not only because it reduces the risk of errors introduced in the development phase but also because it increases the confidence of the customers by providing them with a quality product. Testing a system is the task of validating and verifying if it satisfies the specified requirements and works as expected. It is also the process of running an application with the purpose of finding errors Myers et al. (2011). In simple words, testing an application is executing it following a set of steps and evaluating if its current behaviour is correct compared with the desired result. Software testing is a critical task, but it still has certain limitations as the tests only prove the existence of errors never their absence (Ammann and Offutt, 2008). In an ideal situation, we would test all possible paths that the software has (Myers et al., 2011). However, it is impossible to do an exhaustive test to an application, as even the simplest application may have hundreds or thousands of possible combinations (Myers et al., 2011). Creating test cases for all of these possibilities is impractical.

Software Testers often call a successful test one that finds an error (Ammann and Offutt, 2008). That is why their primary concern is to ensure a good coverage of the tests rather than testing all the possible combinations. Good coverage means that the tests cover the various areas of the application. Concluding, a tested software is not a software without errors, but a software that satisfies the minimum testing guidelines defined by the testers.

2.1 Basic Definitions

To better understand the software testing process and to avoid ambiguities, this section presents some basic definitions. These definitions follow the glossary defined by the *International Software Testing Qualification Board* (ISTQB).

- **Error:** a good synonym is mistake. A mistake made by the developer while coding that produces an incorrect result. It is an incorrect internal state that tends to propagate originating some fault.
- **Fault:** a good synonym is defect. A fault is the result of an error, and is its representation. A flaw in a component or system that can cause the system to fail to perform its required function. A fault, if found during execution, may cause a failure.
- **Failure:** a failure occurs when the code corresponding to a fault executes. It is an incorrect behaviour of a program, a deviation of the system from its expected result.
- **Test:** a test is the act of exercising software with test cases. A test has two distinct goals: to find failures or to demonstrate correct execution.
- **Test case:** a test case has an identity and is associated with a program behaviour. It also has a set of inputs and expected outputs.

2.1.1 Verification and Validation

Software testing can be used for validation or verification. Therefore, when talking about the quality of the software, it is important to distinguish these two concepts.

Validation is the process of evaluating a system to determine if it satisfies the specified requirements. Consequently, validation tests are designed to ensure that the final product meets the customer's expectations (Ammann and Offutt, 2008). Typically, for this type of testing, external sources to the project (such as end users) are used, which allows the testing team to validate the decisions made during the development cycle. For example, tests generated from system requirements are validation tests (Rodrigues, 2015).

On the other hand, verification tests are the process of determining that the developed product remains internally consistent. In other words, ensure that the specifications have been correctly implemented and that the system satisfies these specifications (Ammann and Offutt, 2008). However, they do not guarantee that the system is the one desired by the customer. Verification tests are the most common because they allow comparing the output of the program with the expected output ensuring that the system works as specified.

2.2 Testing Methods

There are several ways to test a system, but they typically fall into two approaches: white-box tests, also known as structural tests, and black-box tests also called functional tests.

The use of one or the other approach depends on the goals to achieve. In a testing process, we can use any of these approaches as they complement each other. However, the results obtained are different since the two methods have different approaches to testing.

The following sections explain these approaches in more detail.

2.2.1 White Box Testing

White-box testing assumes the tester can see inside the "box" (the system) to identify test cases, based on the system implementation (Jorgensen, 2013). The test cases are generated from the source code of the software to cover the implementation to a particular degree (execute every statement in the program, execute all decisions, etc) (Ammann and Offutt, 2008).

This strategy is used to test the system code and verify that it runs as expected. The main advantage of this type of testing is that, as the tester knows the system implementation, the tests are more methodical and cover specific situations becoming more focused on the purpose of the test. The main drawbacks are the costs, as this kind of tests needs to be performed by testers with knowledge about the system implementation.

2.2.2 Black Box Testing

In black box testing, the system is seen as a closed "box" that receives input and produces output (Paiva, 2006). The system is treated as a function that maps domain data to values of the co-domain. To produce test cases the system specification is used, so the generated tests completely abstract the internal behaviour of the system (Ammann and Offutt, 2008).

When applied to GUIs, these tests allow the verification that what is accessible to the user works as expected and meets the requirements and specifications (Rodrigues, 2015). They check whether the specified input generates the expected output, without worrying about how this process is done.

This testing strategy is useful because it allows detecting possible ambiguities and inconsistencies between the system and its specifications. Model-based testing is included in this class of

testing methods because a model is used to generate the test cases, and the defined model is not based on the system implementation, but rather on its specification.

2.2.3 A Comparison of Testing Methods

By the previous description, it is observed that both methods have the same objective, identify and generate test cases. The difference lies in the knowledge used for doing it. As stated earlier, white-box testing uses the source code to generate test cases, while black-box testing uses only the system specification to generate test cases.

Therefore, if the system does not implement all the specifications, white-box testing will never detect this problem. However, if the system implements behaviours that are not specified the black box tests do not detect this behaviour (Jorgensen, 2013). Concluding, both methods described here are important to ensure the quality of the system (Myers et al., 2011).

2.3 GUI testing

Given the increased importance of GUIs, GUIs testing is essential for the proper functioning of a system. However, the increased complexity of the interfaces makes the testing process time-consuming and intensive (Carvalho, 2016).

The methods of testing a GUI are divided into those that require users to use the system (Manual GUI testing) and those that rely on models or simulations of the system for the analysis (Automated GUI testing).

2.3.1 Manual GUI testing

Manual GUI testing consists in the interaction with the application from the user perspective and comparing the results obtained with the expected results. This process can be performed by test engineers or by real users. However, in both situations, there is no guarantee that all features will be covered (Rodrigues, 2015). Considering the high costs of this type of tests, the analysis will not be exhaustive regarding all the possible interactions between the users and the system. Hence, problems with the implementation might remain unnoticed during the analysis. Due to the increasing complexity of GUIs, manual testing tends to be impractical for verification tests, but very useful for validation, like validating if the users find the interface satisfactory. They can

be very useful at an early stage to find errors through end users or trained specialists (Yang, 2011) interacting with early prototypes.

Nevertheless, the creation and execution of manual tests is very laborious and requires much human effort, being a possible source of new errors. Therefore, it is natural that the use of automated testing is increasing.

2.3.2 Automated GUI testing

The scientific community has endeavoured to automate, as much as possible, the process of testing a GUI. Automation allows a reduction in the process' cost and increases the ease of the test cases execution (Ammann and Offutt, 2008).

Automated GUI testing consists of running tests cases that simulate a user's actions on an interface. That is, the tester writes scripts or uses other software to create tests that simulate a user's actions. The created tests allow the tester to verify the application at any time without effort (Farrell-Vinay, 2007). Therefore, this kind of testing is a great help in the development phase, reducing the cost of testing without reducing the number of performed tests.

There are a lot of tools, both research and commercial, that automate this task to varying degrees. These tools range from those that only support the automatic execution of test cases, to those that support generation and execution of test cases.

In this section some common approaches to testing a GUI are described:

- Random Input Testing
- Unit Testing
- Capture/Replay Testing
- Model-Based Testing

Random Input Testing

Random input testing is the simplest testing technique presented here. This technique, also known as monkey testing, aims to perform user actions on a GUI randomly to find errors. The monkeys designation is used to give the idea of "someone" seated in front of a system and interacting randomly with the keyboard or mouse without knowing what he or she is doing (Nyman, 2000). In general, it consists of randomly generating actions (mouse movements, clicks, and keystrokes) without the prior knowledge of the *System Under Test (SUT)* operation. Thus, since there is no

knowledge about the SUT, the execution of this type of tests only allows evaluating if the SUT crashes or freezes (Carvalho, 2016).

Random Testing can be distinguished in two main types, the smart and dumb monkeys. Dumb monkeys have no knowledge of the application other than how to access the GUI. Their behaviour is completely random. As stated by Paiva (2006) the biggest problem with using this approach is that there is no guarantee of the coverage that the tests will achieve. Even more, they do not recognise an error even in the presence of one, which is not very useful.

On the other hand, smart monkeys already have some knowledge of the SUT functioning, generating random actions based on this knowledge. This type of monkeys already detects the presence of errors, being able to determine if the result of an action is correct or not. However, its development is more expensive than the previous ones.

As stated by Yang (2011), although they are useful and allow to detect errors, the coverage of this type of tests is very weak. As the input domain is vast, important actions have a low probability of occurring, so the use of only this test method is inadvisable, being, therefore, more used as a complement to other techniques.

Unit Testing

Unit Testing is a test technique supported by semi-automatic tools that allow programming the test cases. In this testing approach, the test cases are written in a computer programming language which gives a high level of flexibility to the tester (Paiva, 2006).

When applied to GUIs, this type of test consists of the creation of classes or method calls that simulate the interaction of the user with the GUI under test to execute some specific task, while observing the system response to understand if the result obtained is correct.

Typically, because test cases are written manually, the sequence of actions they cover tends to be small, which makes it impossible to discover bugs that occur in specific sequences Yang (2011). So it is likely that there are errors in the GUI that have not been found yet.

Although it is a useful technique and allows to find several errors in the interface, it is a painful procedure to test the GUI conveniently.

Capture/Replay Testing

In this type of software tests, the user's interaction with the GUI is recorded, and the sequence of actions performed is stored in a script. The GUI is tested by running these scripts.

The development of these tests is supported by semi-automatic tools. As the components of a GUI are programmatically readable, these tools enable the testers to create automated tests by

recording the interaction with those elements in a file. The generated files contain a record of all the actions performed, including mouse movements, chosen options, entered inputs and the obtained result. These files can be edited manually but as they were not intended to be 'human readable' and have a language dependent on the tool used, their maintenance becomes quite difficult to the tester (Paiva, 2006).

Although this type of test automates the capture and execution of test cases, they have some disadvantages. If any component of the interface changes, the tests have to be re-recorded. This, combined with increasingly sophisticated GUIs, makes the process of composing a set of tests very arduous (Blackburn et al., 2004). However, the main drawback is that the design of the tests is still done manually. Such as stated by Yuan et al. (2011), in this type of approach the quality of the tests and their coverage is directly related to the competence of the software tester.

Model-Based Testing

In the MBT approach, the test cases are generated from a model (an oracle) that describes the states and events of the GUI under test. The produced tests are executed and the produced output compared to oracle's prediction (Blackburn et al., 2004).

Therefore, this technique is used to test the conformity between an implementation and its model (Yang, 2011). However, to use this approach knowledge of the system and its specification is required. In the next section, this testing approach is presented in more detail.

2.4 Model-Based Testing

MBT is a black-box test technique that allows automating the process of generation and execution of test cases to GUIs, at a much lower cost than manual techniques. The high differentiation is that instead of writing the tests manually, based on the requirements and specifications of the system, a model that represents the expected behaviour of the SUT is created. The model is an abstract and straightforward representation of the desired behaviour of the system (Utting et al., 2012). As the system model is developed based on its specification the purpose of these tests is check if its implementation is the specified.

The basic idea is to construct an abstract model that represents the behaviour of the SUT and use this model to generate automatically a huge number of test cases. After the test generation, the built model is used as an oracle (Silva et al., 2008). The discovery of system errors is achieved by comparing the results of the test execution and the artefact that specifies the desired behaviour of the system (oracle).

With this kind of approach, a high level of test automation can be accomplished, as the test cases generation, test cases execution and the comparison of the oracle's prediction with the real results can all be automated (Paiva, 2006).

2.4.1 The process

The MBT process (see Figure 1) begins with the development of the abstract model of the SUT. Obviously, the developed model must be validated, so it is assumed that the model is simpler than the SUT, or at least easier to validate and maintain. Otherwise, the effort of validating the model will be equal to the effort of testing the SUT (Utting et al., 2012).

The next step is generating test cases from the developed model. The tests generated in this phase are sequences of operations expressed on the model. To decide the number of tests generated it is usual to define a coverage criterion over the model, and the generation process is concluded when this coverage is achieved (Paiva, 2006).

Given that the generated tests are still sequences of abstract actions on the model, it is necessary to convert them to concrete test cases, enabling their execution in the SUT. Finally, the tests are executed, and the analysis of the results is performed, verifying if the results are the expected.

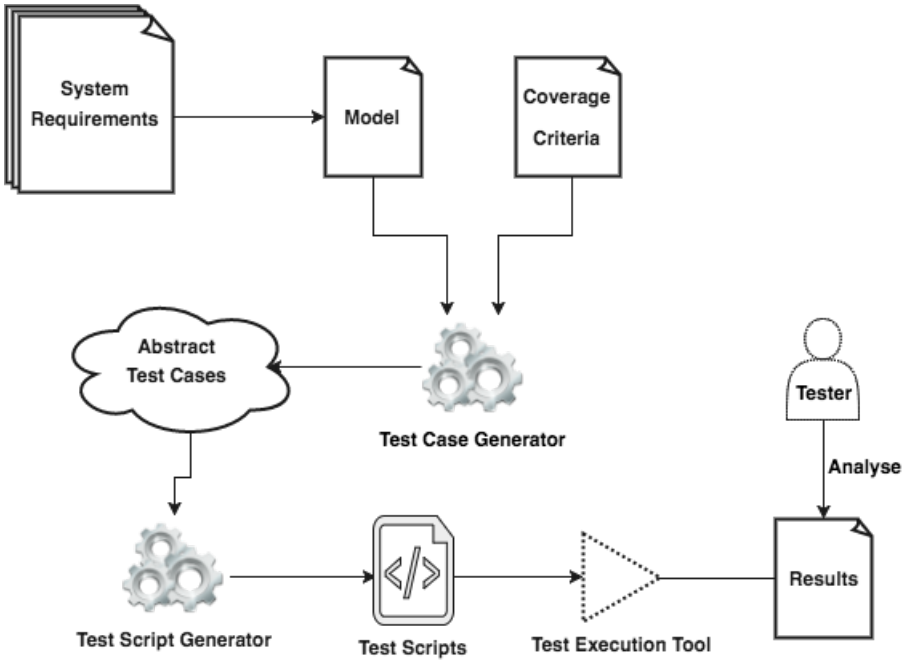


Figure 1.: The Process of Model-Based Testing (Adapted from Utting et al. (2012))

2.4.2 Model-Based Testing of User Interfaces

Model-Based Testing can be used on Graphical User Interface to test if the developed user interface matches its specification. Several projects have been drawn up to allow the use of MBT in GUI testing. These projects mostly focus in automating the process of testing. Nevertheless, work on the application of MBT to GUIs has addressed several areas. Two particularly relevant ones are which models to use as oracles (and how to obtain them) and how to guarantee a quality test suite.

Following are some approaches to the application of MBT to GUIs.

GUITAR

The first project developed in this area was by Memon (2001), who presented the GUITAR GUI testing framework to automate GUI testing, based on a reverse engineering process. Built on a modular architecture, this framework has a set of tools that allows the use of model-based tests on GUIs (See Figure 9). Using GUITAR, the tester can generate and execute test cases, verify the correctness of the GUI and obtain coverage reports from the test execution.

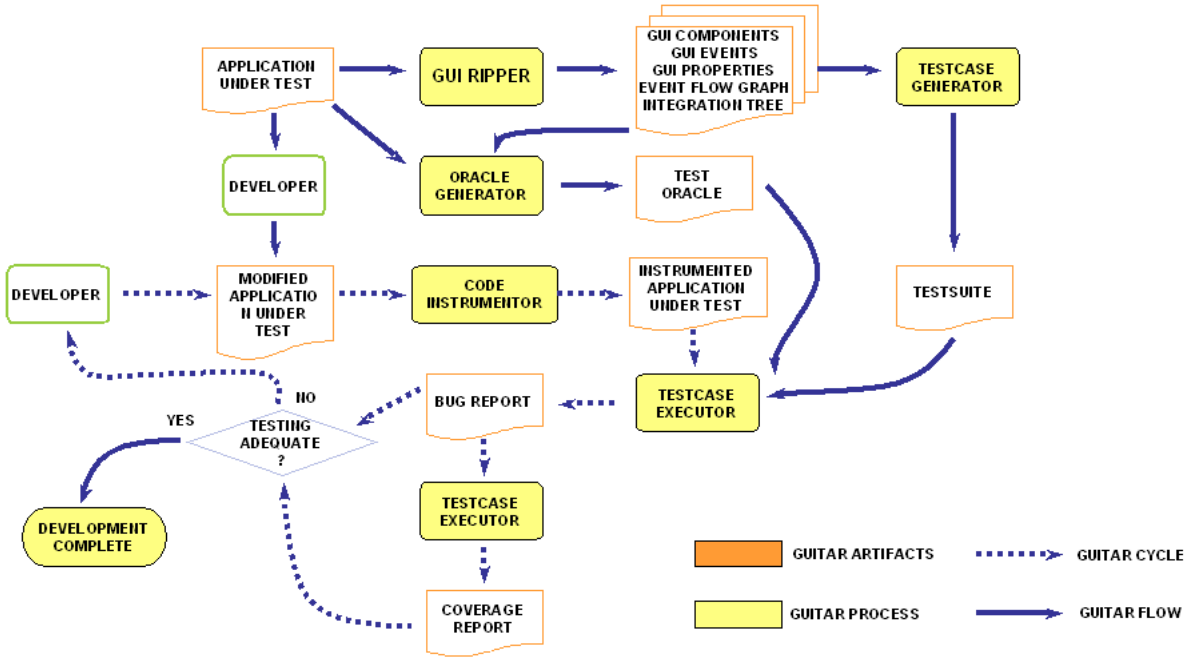


Figure 2.: GUITAR Process (Taken from https://www.cs.umd.edu/~atif/GUITAR-Web/guitar_process.htm)

This tool applies a reverse engineering process to generate the representation of the GUI as an integration tree and an event-flow graph, reducing the time and effort required to build the model. From these artefacts, test cases are generated to test the GUI. The integration tree represents the way the GUI components are integrated to form the complete GUI. It is based on the application's GUI hierarchy and is obtained through the GUI Ripper tool (Memon et al., 2003). In this artefact, each node of the tree represents one GUI component. Event-flow graphs represent all possible interactions among the events in the GUI component and are obtained through the event-flow graph generator. The generated models are easily analysed by their graphical visualisation. Despite this, the structure of the models is not well documented, and the models cannot be built or edited manually, this process is fully automatic and assured by the tool.

The GUITAR tool supports the generation and execution of three types of test cases, structural, random and manual tests. After running the tests the tool provides a report which indicates the GUI components tested and the coverage reached by the test.

Spec Explorer

Spec Explorer[1] is a Microsoft Research tool that supports the application of MBT to native applications. This tool works as an extension of Microsoft Visual Studio, and the main goal is creating models of software behaviour, analysing those models with graphical visualisation, checking the validity of those models, and generating test cases from the models. This tool is suitable for testing *Application Program Interfaces (API)* but requires much effort to be applied to GUI testing. Faced with this difficulty, some extensions were developed.

Paiva (2006) developed an add-on for Spec Explorer, the GUI Mapping Tool, to adapt the tool to perform tests on graphical user interfaces. The GUI Mapping tool tries to automate the process of mapping between the elements of the model and the implementation, helping the user to relate logical actions of the model with physical actions in the GUI (Paiva et al., 2005). In the end, the mapping information is generated automatically.

In this work, test methods based on formal specifications are used to systematise and automate the GUI test process. However, the use of Spec# as the GUI modelling language means that the developers would have to learn this language to adopt this proposal (Paiva et al., 2005).

PBGT - Pattern-based GUI Testing

Another approach developed by Moreira and Paiva (2014) is *Pattern Based GUI Testing (PBGT)* based on the UI Test Patterns concept that presents a set of test strategies that simplify testing

1 https://msdn.microsoft.com/en-us/library/ee620411.aspx

of UI Patterns on GUIs. This approach aims to promote the reuse of test strategies to test typical behaviours in web applications.

PBGT is supported by a Eclipse plugin developed on top of the Eclipse Modelling Framework, the PBGT Tool. The tool provides an integrated modelling and testing environment that supports the crafting of test models based on UI Test Patterns, using a GUI modelling *Domain Specific Language (DLS)* called PARADIGM. UI Test Patterns are elements within PARADIGM from which it is possible to build models describing the GUI testing goals at a higher level of abstraction. The models generated in this environment are used to generate and run test cases over a GUI.

So, this technique takes advantage of UI patterns to test the GUI more efficiently. However, the tool is limited to the number of UI Test Patterns supported, and to the fact that UI patterns must be found in the GUI under test.

TOM Framework

The TOM Framework is under development in the HASLab Laboratory of the University of Minho. It is a set of tools designed to promote the introduction of advanced model-based testing technologies in the process of developing interactive software applications.

At the beginning of this dissertation, the Tom Framework consisted of two components: the TOM Generator, and the TOM Editor.

TOM Generator:

TOM Generator contains the core of the TOM Framework. This component has a modular architecture to ensure the adaptability of the framework and includes modules to read and translate different types of models to its internal representation, and modules to generate, mutate and execute the test cases. TOM Generator has been designed aiming at a low maintenance cost, and its adaptation to different contexts, so the addition of new modules and the change/replacement of existing one has a limited impact on the system.

The TOM Generator developed by Rodrigues (2015), was a Java Desktop application without an interface. That is, to use the capabilities of the application it was necessary to change the source code to specify the directories of the files (models, mappings, mutations and values) and the parameters desired for the generation (such as algorithms, mutations to generate, browser to use, ...). After this changes, the code was compiled and ran. The tests were generated to the specified directory.

Despite the fact that the target audience of the application is composed of people with technical knowledge in the area, the fact that the end user can view/change source code is not at all a

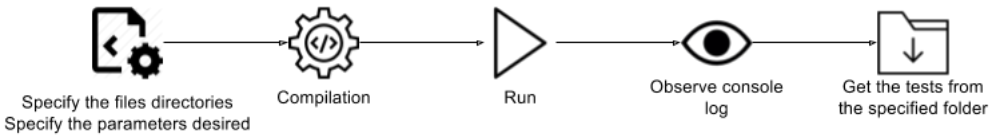


Figure 3.: TOM Generator: Operating Mode

good option for an application. Thus, this component has undergone many changes throughout this dissertation, being these presented in Chapter 4.

TOM Editor:

The TOM Editor, developed by Pinto (2017), is a browser extension that supports the development of a model of a web application GUI. The TOM Editor allows capturing the user interaction with a web page while simultaneously defining the system model based on the user interaction. Through the TOM Editor, the model of a GUI can be constructed in a semi-automatic way.

The purpose of the Editor is to facilitate the creation of the model preventing the user from making modelling errors (Pinto et al., 2017).

Other Approaches

Other authors have explored different approaches to reduce the production cost of the models. Silva et al. (2008) use a task model as an oracle. Task models are widely used to represent the tasks the user can perform in a GUI. In this approach a task model of the SUT is used to generate a state machine. This is achieved through the TERESA tool[2], which generates a representation called Presentation Task Sets (PTS). From this representation and the mapping of the model to the real interface, a graph is created from which the test cases are generated. As the task models only represent the correct behaviour of the user, the use of mutations in the test cases was also explored (Cruz and Campos, 2013), allowing the tests to cover variations such as user's errors.

In addition to these approaches, many other authors have focused on integrating model-based testing with graphical user interface testing. Lelli et al. (2015) developed an approach that focused not on the cost but on the expressiveness of the model by proposing a modelling language capable of dealing with advanced multi-event GUIs. Bowen and Reeves (2011) explore the applicability of test-first development, an approach similar to test-based development, but using models of the requirements as the basis for the tests, to GUI development.

2 http://giove.isti.cnr.it/teresa.html

2.5 Summary

In this chapter, the thematic of software testing and its importance was introduced. A system must be validated and verified to increase our confidence in its quality. With validation, it is checked whether the system satisfies the requests of the users. With verification, it is investigated whether the system was correctly implemented. Both qualities can be improved through software testing, however, although it does not guarantee the complete absence of errors.

GUI testing is an essential component of the interactive software development process. In this chapter, the different approaches to GUI testing were presented. There are several procedures for testing a graphical user interface. Typically the different procedures are divided into one of two kinds of techniques to test the graphical user interface. One focused on the usability of the GUI, where the interest is the opinion and the difficulties of the users, and another more focused on the implementation, where the importance is to ensure the system quality, both in behaviour and in performance.

This chapter addressed model based testing, a technique that focuses on the quality of a software implementation. Model-based testing is a technique that allows automating the process of generation and execution of test cases to GUIs, at a much lower cost than manual techniques. Some MBT tools that allow the use of the MBT to the GUIs were presented. The application of MBT to GUIs permits the increase of software quality, reducing the time and effort spent to develop the tests. The high degree of automation provided by this technique allows the testing of GUIs more exhaustively and reliably. However, the success of this technique relies on the model and the quality of the tool used to obtain the test cases.

Here we describe the development of the TOM Framework that aims to provide a more flexible and adaptable support for the application of the MBT to the GUIs (Rodrigues, 2015). TOM is similar to GUITAR as it adopts a plug-in architecture to provide flexibility and extensibility. However, while GUITAR is based on reverse engineering the SUT for testing. TOM adds a focus on flexibility and also on the modelling side.

Chapter 3

TOM Generator

In this chapter, one of the components of the TOM Framework, TOM Generator, is presented. The improvements made to the generation process are also described.

TOM Generator is a crucial component of the TOM Framework. It consists of a set of flexible and adaptable modules that aim to facilitate the application of MBT on GUIs.

3.1 Generation process

The test case generation process used by the TOM Generator component is divided into several steps, following the typical MBT approach, explained earlier (Subsection 2.4.1). Figure 4 illustrates the test case generation process used.

The component, after reading the GUI model, represents it as an oriented graph. With this representation, abstract test cases (test cases that are independent of a concrete implementation) are just paths in the graph. To obtain concrete test cases, after running the traversal algorithm, the abstract test cases must be converted to source code in the required language. After obtaining the concrete test cases, these are exported to executable test files.

Converting abstract test cases to concrete test cases requires the user to provide some settings, such as the type of tests to be generated and the mutations to be introduced. Currently, the framework has three modules to generate test cases, a module that allows the generation of web interfaces tests, which later may be run by Selenium web-driver, another that creates scenarios in Hamsters' notation and the last one that generates test cases in *JavaScript Object Notation (JSON)* notation. Depending on the type of tests to generate and the mutations to be introduced, some configuration files may be needed to complete the model (Values, Mutations and Mappings).

In summary, the TOM Generator process is divided into seven stages:

1. Reading and translating the system model
2. Generating a graph from the system model

3. Application of traversal algorithms on the previously obtained graph, generating abstract tests cases
4. Converting abstract test cases into concrete test cases; Additionally, mutations can be introduced
5. Creating the files with the executable test cases
6. Executing the tests

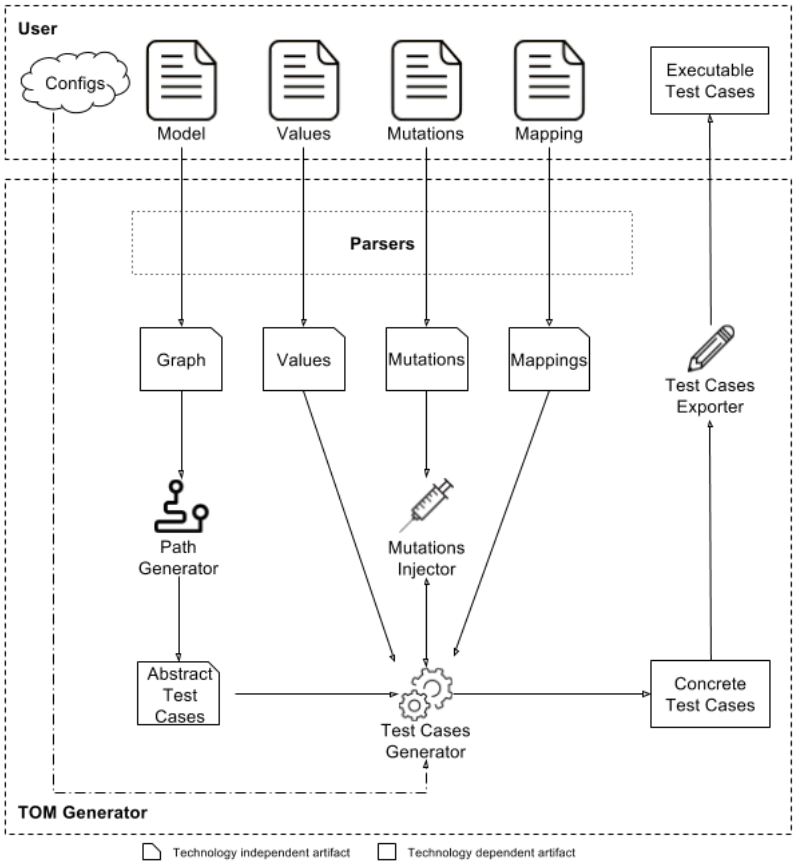


Figure 4.: Generation process

The format of the files necessary for the generation of test cases are presented below. The files presented are based on the generation of tests for web applications. However, the only difference is that for the other types of tests currently supported the mapping file is not necessary because in that case the tests are generated at an abstract level.

3.1.1 Modelling the user interface

To represent a GUI the TOM Framework uses *Final State Machines (FSM)* as they are a recurring approach to modal behaviour in software engineering.

In the representation of a graphical interface through a state machine, each state represents a dialogue window or a page in a Web application. In each state, there may be transitions (links, buttons, ...) to new states, which represent changes in the interface as a response to the user, or several sub-states (form, menus, ...) (Rodrigues, 2015). For the model to be read by the tool, it is necessary that the state machine is represented in one of the supported formats. Currently parsers for *State Chart XML (SCXML)* and Emucharts are available. In this section, only the representation of the state machine through a SCXML file is presented.

Listing 3.1 is an example of a model. In the presented excerpt a state can be observed (from line 3 to line 16), which is how a web page is represented. The presented state has a sub-state (from line 6 to line 15) representing the filling of a form. The form has three input fields represented by the "send" tag (lines 7, 8 and 9) and a transition to a new state (from line 10 to 13). In case of success, the "submit" field from the transition is used (line 11), in case of error the "error" field (line 12). There are also two validations (line 4 and line 14), represented by the "onentry" and "onexit" tags. In case of non-completion of the form, the application can still transit to another state through the state transition (line 5). The "id" and "label" attributes identify concrete values defined in the mapping and value files, these values are used to complete the tests.

```
1  <scxml version="1.0" ... name="Editor" initial="20161141559010">
2
3    <state id="20161141559010">
4      <onentry id="2017031127001" type="displayed?" />
5      <transition id="20161141559012" target="20161141559011" />
6      <state id="sign in" type="form">
7        <send label="s20161141559032" type="required" element="checkbox" />
8        <send label="s20161141559033" type="required" />
9        <send label="s20161141559035" type="required" element="checkbox" />
10       <transition type="form" label="20161141559037">
11         <submit target="20161141559038" />
12         <error target="20161121851003" />
13       </transition>
14       <onexit id="2016114208001" type="contains" />
15     </state>
16   </state>
17 ...
18 </scxml>
```

Listing 3.1: Example of Model file

The construction of the SCXML GUI model and the description of the elements used to describe the interaction and validation of the GUI are explained in detail by Pinto (2017).

Mappings and Values

During the development of the model, there are elements (such as concrete values for input fields, or mappings between the model and *Hypertext Markup Language (HTML)* elements) necessary to create executable tests that are not represented in the state machines, because we want to keep the model abstract. Attributes are introduced (such as labels or ids) allowing us to map from the abstract level to the concrete level. The separation between states machine, mappings and values was done to make the models more general and easy to read

To associate the attributes entered in SCXML, the HTML element and the input data for forms and validations, two types of files are used. To associate the HTML element of the SUT and the corresponding attribute in the state machine, a mapping file is used. A values file is used to fill in the form data information and the validations. The rules for building both files are explained by Pinto (2017).

An example of a mapping file is shown in Listing 3.2. This file is used to map the behaviour defined in the state machine with the WEB application. For example, the element with the label "s20161141559033" (from line 7 to 11) of the model will be found through its name (line 8), its name is "user[email]" (line 9) and the test must must fill in the text field ("sendKeys" line 10).

```
1  {
2    "2017031127001": {
3      "how_to_find": "className",
4      "what_to_find": "header-full-title",
5      "what_to_do": "getText"
6    },
7    "s20161141559033": {
8      "how_to_find": "name",
9      "what_to_find": "user[email]",
10     "what_to_do": "sendKeys"
11   },
12     "s20161141559035": {
13     "how_to_find": "className",
14     "what_to_find": "translation_missing",
15     "what_to_do": "click"
16   }
17 ...
18 }
```

Listing 3.2: Example of Mapping configuration file

An example of a values file is displayed in Listing 3.3. This file is used to define values to enter in the test cases. Resuming the previous example, the text to be sent to the element "s20161141559033" will be "admin@admin" (line 3).

```
1 [
2   { "2016114208001" : "successfully" },
3   { "s20161141559033" : "admin@admin" },
4   { "s20161141559035" : "" },
5     ...
6 ]
```

Listing 3.3: Example of Values configuration file

The values defined in this file are used in test cases to fill in forms or to perform validations. In the form filling the values are entered in the input fields to simulate a user, in the case of the validations, the values present in the file are compared with those present on the HTML during the execution of the tests.

Mutations

The model used will typically represent the expected behaviour of the user. Hence, the test cases generated directly from this model follow the path that the user should use. In other words, the tests cases perform actions according to what is specified. This kind of tests is important because it allows us to ensure that all relevant functionalities are tested. However, unpredictable behaviours of the user are not covered.

Although positive tests are essential, they need to be complemented with tests that represent the unexpected behaviour of the user (Rodrigues, 2015). To obtain this kind of tests, and thus increase the quality of the tests generated, the TOM framework introduces mutations in the test cases. Mutations are controlled changes to the test cases that introduce small errors in the test, to see how the application reacts to the unexpected user behaviour. The purpose of these mutations is to run the tests in a 'faulty' way and cause the test to fail, increasing the reliability and quality of the software.

The TOM Generator supports the mutation of test cases in a random way or through a configuration file, in this file, the type of mutation to perform and the action where it is expected to be performed can be specified (Pinto, 2017). This file can be generated from the TOM Editor, or manually constructed, Pinto (2017) explains how to build this file to model the mutations to perform in the tests. In Listing 3.4 an example of a configuration file for the mutations can be seen. Analysing the file example, the element "s20161141559035" of the model (line 4) should undergo a "lapse" mutation (line 3), and this mutation does not cause the test to fail (line 5).

```
[
  {
  "type" : "lapse",
  "model_element": "s20161141559035",
  "fail" : "0"
  },
...
]
```

Listing 3.4: Example of Mutation configuration file

The TOM Generator is capable of generating several types of mutations. However the mutations generated depend on the type of tests to be generated, in the next section the mutations supported by each type of test are explained.

3.2 Type of generated tests

As previously mentioned, after reading and interpreting the files that model the system, the TOM Generator internally performs several steps to generate the executable test cases. Initially, abstract (technology-independent) test cases are generated, and then the abstract test cases are converted into concrete test cases, and finally to executable test cases.

Currently, the TOM Generator is adapted for the generation of three types of test cases:

- Web Applications: structuring actions in Selenium instructions, using annotations from TestNG
- IRIT Scenarios: contains a list of steps to go through during the simulation in CIRCUS that when co-executed with the task model detect if there is any deviation in expected behaviour
- JSON: contains a list of steps to go through the interface

In the following sections each of the types of tests supported by TOM Generator is presented.

3.2.1 Web Applications

The generation of tests for Web Applications was the first type of test supported by TOM Generator, its development was started by Rodrigues (2015) and is improved in this dissertation.

The test cases are generated in Java and run using the Selenium and TestNG/JUnit frameworks. Two frameworks are used because they complement each other. Selenium connects with

the browser and allows web applications to be tested on the browser in an automated way. TestNG/JUnit organises the tests and creates reports more easily.

- **Approach**

As previously mentioned, the TOM Generator allows the user to configure the generation of test cases according to their needs. To generate such test cases it is necessary to provide the following configurations:

- Algorithm and stopping criterion to be used
- System Model (State Machine)
- Mapping File
- Mutations to generate (optional)
- Values File (optional)
- URL of theSUT (optional)

The algorithm to be used and the stopping criterion are common to all test types as well as mutations and the values file. The unique settings for Web App testing are the mapping file (explained above) and the URL that indicates the address of the application being tested.

In this type of tests, the test case generation algorithm runs through each step of the abstract test, generating the corresponding Java code. Rodrigues (2015) decided that for each abstract test a group of tests was created (one for each test step), rather than the entire sequence of test steps in a single method. This decision was taken to follow good software testing practices to create small, reusable test cases. However, as found by Pinto (2017), when one of the tests in the group fails, the rest of the tests in the group still run, which means that the tests dependent on the failed test also fail.

The fact that several tests fail rather than a single test leads to the need to analyse test to test to find out the reason for the failure. Thus, it was decided to change this behaviour so that an abstract test generates a single test instead of a group. With this change, when a test fails it is easy to know where the error occurred, and it is not necessary to analyse if the error comes from a previous failed test. With this change, the number of tests generated has decreased significantly. However, it is guaranteed that the tests generated by the TOM Generator before the change and after the change are similar and guarantee the same coverage of the SUT, since the test groups are aggregated in a single test case.

In Listing 3.5 there is a test generated after the changes, the abstract test that originated the test presented would originate nine tests before the changes. However, the test result would be the same.

```
1   /** Path number: 1 Path: [9010]  -> CALL 9012, [9011]  -> */
2     @Test(invocationCount = 1, groups = "1")
3     public void test1() throws IOException, InterruptedException {
4         ...
5         // page 9010
6         Reporter.log(" Enter in page 20161141559010 <br>");
7
8         // Asserts
9         try { ... } catch (AssertionError _x) { ... }
10
11        // Validation Test On entry in a Page
12        try { ... } catch (AssertionError _x) { ... }
13
14        // click 9011
15        try { ... } catch (WebDriverException _x) { ... }
16
17        // Asserts
18        try { ... } catch (AssertionError _x) { ... }
19    }
```

Listing 3.5: Example of a Java test case structure

- Mutations

As mentioned, the TOM Generator supports the mutation of test cases to introduce common use errors in the tests generated. Next, the mutations supported by this type of tests are presented.

- Slips, the order of execution of the actions is changed
- Lapses, one of the actions of the test is deleted
- Mistakes, a value in a form is changed
- Remove required field, some required fields are removed from the test
- Double clicks, a double click is made on all buttons used during the test
- Pressing the back button, a back event is injected at the end of each step in the test
- Refreshing the page, the page is refreshed at the end of each step in the test

All of the above mutations can be introduced randomly, or we can specify the mutations that we wish to introduce through the mutation file. Through this file, greater control of the mutations can be achieved, as we can decide where the mutations are introduced. Thus, specific situations of the application can be tested. In the case of random mutations we have no control, and from generation to generation, the mutations are introduced at different occasions. With the

introduction of the mutations in the test cases, the most common mistakes made by users when interacting with WEB applications can be covered.

During this dissertation, the introduction of mutations was revised. When mutations could not be introduced in the test cases, the original test without mutations were being regenerated, resulting in numerous repeated tests. Thus, the way the mutations were introduced was changed so that if introducing mutations in the test is not possible another test without mutations is not generated .

3.2.2 IRIT Scenarios

The generation of scenarios resulted from a partnership with IRIT, where it was intended to adopt the TOM Generator so that it could generate test scenarios based on HAMSTERS task models (Martinie et al., 2011). The adaptation of TOM Generator was initiated by Pinto (2017), during this dissertation the partnership was continued to improve the generation of scenarios, namely to improve the generation process and to allow the introduction of mutations and values in the scenarios generated.

In this type of tests, the TOM Generator is used to generate scenarios. The scenarios are generated as XML files and executed in CIRCUS (Fayollas et al., 2016), a tool supporting task modelling and analysis in HAMSTERS. The generated scenarios are used in IRIT to verify the interface under test.

- **Approach**

The TOM Generator adaptation allowed the application of model based testing techniques to task models developed by IRIT. Hamsters task models define the possible sequences of user and system actions to achieve a goal. Tasks can be of several types (see Figure 5) and contain information such as a name, information details, and criticality level. Only the single user high-level task types are presented.

Task type	Icons in HAMSTERS task model
Abstract task	Abstract task
System task	System Task
User task	User Task, Perceptive Task, Cognitive Task, Motor Task
Interactive task	Interactive Input Task, Interactive Output Task, Interactive input output task

Figure 5.: High-level Task Types in HAMSTERS (Adapted from Campos et al. (2017))

HAMSTERS supports hierarchical task modelling. Hence, a HAMSTERS task model is a decomposition of a root goal into a tree of sub-goals and atomic activities. Temporal operators (presented in Table 1) are used to represent temporal relationships between sub-goals and between activities. Tasks can also be tagged by temporal properties to indicate whether or not they are iterative, optional or both.

Operator type	Symbol	Description
Enable	T1 >> T2	T2 is executed after T1
Concurrent	T1 || T2	T1 and T2 are executed at the same time
Choice	T1 [] T2	T1 is executed OR T2 is executed
Disable	T1 [> T2	Execution of T2 interrupts the execution of T1
Suspend-resume	T1 |> T2	Execution of T2 interrupts the execution of T1, T1 execution is resumed after T2
Order Independent	T1 |=| T2	T1 is executed then T2 OR T2 is executed then T1

Table 1.: Temporal Ordering Operators in HAMSTERS (Adapted from Campos et al. (2017))

In Figure 6 an example of a task model is presented. At each instant, a set of actions is allowed. Executing an action, changes the set of allowed actions, as defined by the task model hierarchical structure and operators. The set of task available at a given moment is called a *Presentation Task Set (PTS)*.

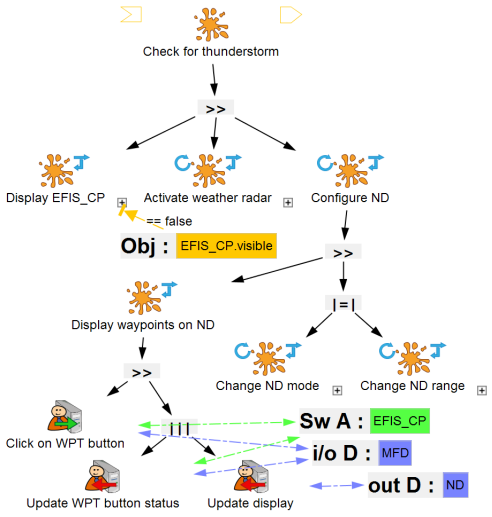


Figure 6.: Task model representation

Using the PTS, a state machine (Model) is developed from the task model representing its behaviour. The state machine is (currently) developed manually, considering the PTS as states in the state machine and labelling the transitions with the tasks. The algorithm starts by calculating the initial PTS of the model (which constitutes the initial state) and then, for each task in the

PTS, calculates the PTS/state resulting from its execution (Campos et al., 2017). The algorithm is applied to all PTS until all tasks in all PTS have been exercised.

After the state machine is created, it is used to generate the list of abstract scenarios that will be used to automatically generate the test cases. Figure 7 presents the whole process applied to the generation of scenarios, from the conversion of the task model to the execution of the scenarios. The orange rectangle is the competence of the TOM Generator.

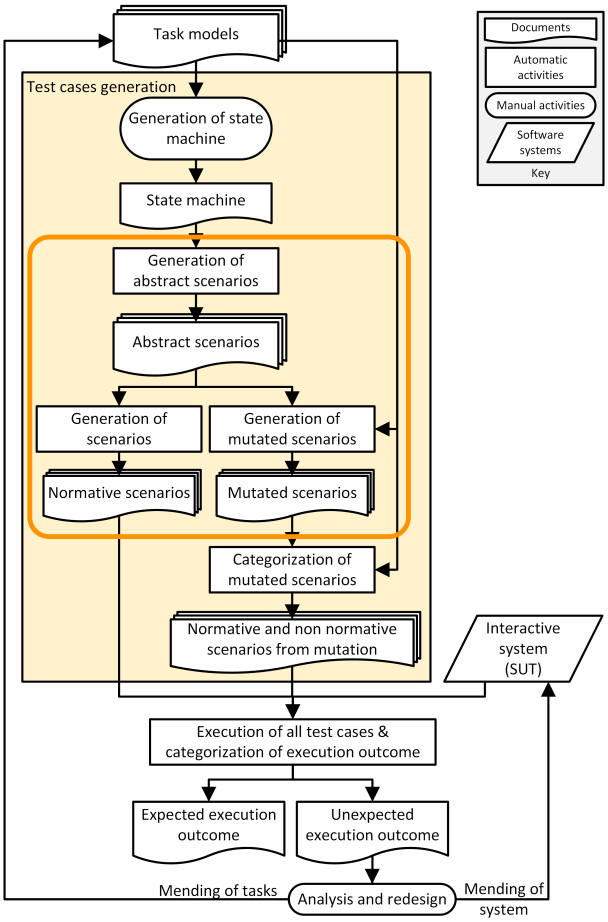


Figure 7.: Scenario generation process (Campos et al., 2017)

In order to use the TOM Generator for the generation of test scenarios it is necessary to provide the following configurations:

- Algorithm to be used and stopping criterion
- System Model (state machine)
- Mutations to generate (optional)
- Values File (optional)

Scenarios are generated as XML files so that they can be executed by IRIT in CIRCUS. An example of a scenario generated by TOM Generator can be found in Listing 3.6.

```
<hamsterscenario date="0" simulatedmodel="Model" version="0">
  <objects>
    <object objectClass="int" objectContent="1" objectID="headingEditbox"/>
  </objects>
  <steps>
    <step referencemodel="Model" role="pilot" taskdate="0" taskdatelong="0">
      <task taskid="t447">
    </step>
    <step referencemodel="Model" role="pilot" taskdate="0" taskdatelong="0">
      <task taskid="t435">
        <stepObject objectID="headingEditbox"/>
      </task>
    </step>
  </steps>
</hamsterscenario>
```

Listing 3.6: Example of a generated scenario

- **Mutations**

During the realization of this dissertation the ability to introduce mutations in the scenarios was introduced in the IRIT scenarios generation component. So, the scenarios generated can be: normative or mutated. Normative scenarios capture concrete sequences of action as described by the task model. Mutated scenarios intent to capture possible user errors as deviations from the norm (described by the task model).

Mutations can be introduced in a randomised way or through a mutation file previously explained. In the generation of scenarios, the following mutations can be introduced:

- Slips, the order of execution of the tasks is changed
- Lapses, one of the tasks of the test is deleted

3.2.3 Json

The generation of tests in JSON is the latest adaptation of the TOM Generator. The development of this adaptation was started during this dissertation. Nevertheless, it is still in an initial phase.

This adjustment derived from the attempt at generating test cases for PVSio-web[1], a graphical environment for facilitating the design and evaluation of interactive systems. PVSio-web can generate and evaluate realistic interactive prototypes from formal models and has been successfully used for the analysis of commercial, safety-critical medical devices.

The purpose of this adaptation is to allow the generation of test cases to automatically test emulated medical devices through PVSio-web.

- **Approach**

PVSio-web allows the creation of models of the devices through the EmuCharts Editor (Masci et al., 2015). The EmuCharts Editor implements the visual editor for the creation of emucharts of medical devices. An Emucharts diagram is the representation of a state machine in the form of a directed graph composed of nodes and transitions (Mauro et al., 2016).

The generation of test cases for PVSio-web was achieved through the adaptation of TOM Generator for the reading of emucharts and for the generation of test cases in JSON. Thus, two new modules were created in TOM Generator, one that supports the reading of emucharts and a second that supports to generate the test files in JSON.

```
[
  {
    "id": "LoadSyringe",
    "name": "INIT_LoadSyringe",
    "ts": 1487359452533
  }, {
    "id": "IInit0",
    "name": "IInit",
    "ts": 1487359452535
  }, {
    "id": "IInfo6",
    "name": "IInfo",
    "ts": 1487359452536
  }
]
```

Listing 3.7: Example of a JSON test case

The test generation process is quite similar to the previous ones. After generating an emuchart through PVSio-web, the emuchart is imported into the TOM Generator and translated into its internal representation through the new added module. After reading the emuchart, the normal

1 http://www.pvsioweb.org/

generation process is applied and the JSON generation module is used to generate the test cases. An example of a test generated for this type of test is shown in Listing 3.7.

As already mentioned this adaptation is still at an early stage of development, so it is still necessary to improve some generation processes, such as the inclusion of input values or the support for mutations of test cases.

3.3 Summary

In this chapter the capabilities of the TOM Generator were presented. TOM Generator is a tool capable of generating multiple types of test cases.

All improvements made to the test generation were presented. Namely, in the tests generated for web applications, the test groups were united in a single test, the mutations were reviewed and the repeated tests eliminated. In the scenarios generated for IRIT, the ability to introduce mutations in the scenarios, as well as values in the input fields, was introduced. Finally, the TOM Generator was adapted to generate JSON tests.

Chapter 4

Contributions

In this chapter all the contributions made throughout this dissertation are explained in detail. Firstly, the improvements made to the TOM Generator component to improve the performance of the Framework are described. Then, the development TOM App is presented.

To support TOM Generator integration with other tools (such as IRIT's CIRCUS[1]), it was decided to improve the component's architecture, by developing a Web Services layer, so that access to its functionalities could be obtained through requests to a server. The only constraint imposed was that the modular architecture present in the application remained so that editing, adding, and removing modules would not have much impact on the system.

4.1 Code Refactoring

As previously mentioned, TOM Generator was developed based on a modular architecture, so that it can be easily adapted to different contexts. Thus, one of the main concerns in this dissertation was to maintain this architecture.

After an analysis of the existing code, it was concluded that there are three types of modules crucial to the adaptability of the TOM Generator. The first type of module are the Algorithms. Test cases are generated resorting to graph traversal algorithms, so maintaining and adding algorithms to the framework should be easily done. The second type are the Parsers. As the test cases are generated from models, it is crucial that the TOM Generator can generate tests from different kinds of models, so it is necessary that it can read the different kinds of models into its internal representation. Finally, the Generators are modules that transform abstract test cases into executable scripts. In a world where technologies change from day to day, it is essential that TOM Generator be able to generate software tests regardless of the technology used.

Despite the existing modular architecture, no rules were defined regarding the addition of new modules, making it necessary to change much code to use them. Additionally, it also meant that

1 https://www.irit.fr/recherches/ICS/softwares/circus/

it was impossible to standardize how modules should be used, which made it impossible to let a user choose which module to use at runtime, a requirement for the introduction of the Web Services layer in the TOM Generator.

Thus, to introduce some rules in the creation of modules, and to increase the ease of maintenance and addition of the modules, some Design Patterns have been applied to the TOM Generator's architecture in order to restructure it.

4.1.1 Patterns Used

Concerning the Parsers, the Strategy pattern was applied. The Strategy pattern defines a set of encapsulated algorithms that can be swapped to carry out a specific behaviour (Gamma et al., 1995). In Figure 8[2] it is possible to observe the application of the pattern in the parsers of the TOM Generator.

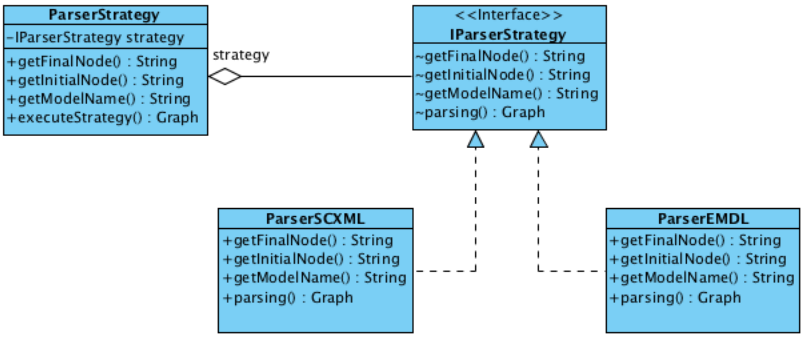


Figure 8.: Strategy pattern applied to the parsers

The goal of using the Strategy pattern is to define a family of parsers, to encapsulate each one of them, and to make them reusable in more than one situation. As can be seen in Figure 8 there are two parsers in TOM Generator (ParserSCXML and ParserEMDL), these parsers support two types of models. If we want to add support for a new kind of model, we need only to add a new parser that implements the IParserStrategy interface and configure the TOM Generator for the possibility of using this new parser. After applying the pattern, the choice of which parser to use is left to the client, so different clients can use different parsers.

For the case of the Algorithms and the Generators modules, the Abstract Factory pattern was applied as shown in Figure 9. The Abstract Factory provides an interface for creating families of related or dependent objects without specifying their concrete classes (Gamma et al., 1995).

2 The parameters of the methods were all removed for ease of presentation.

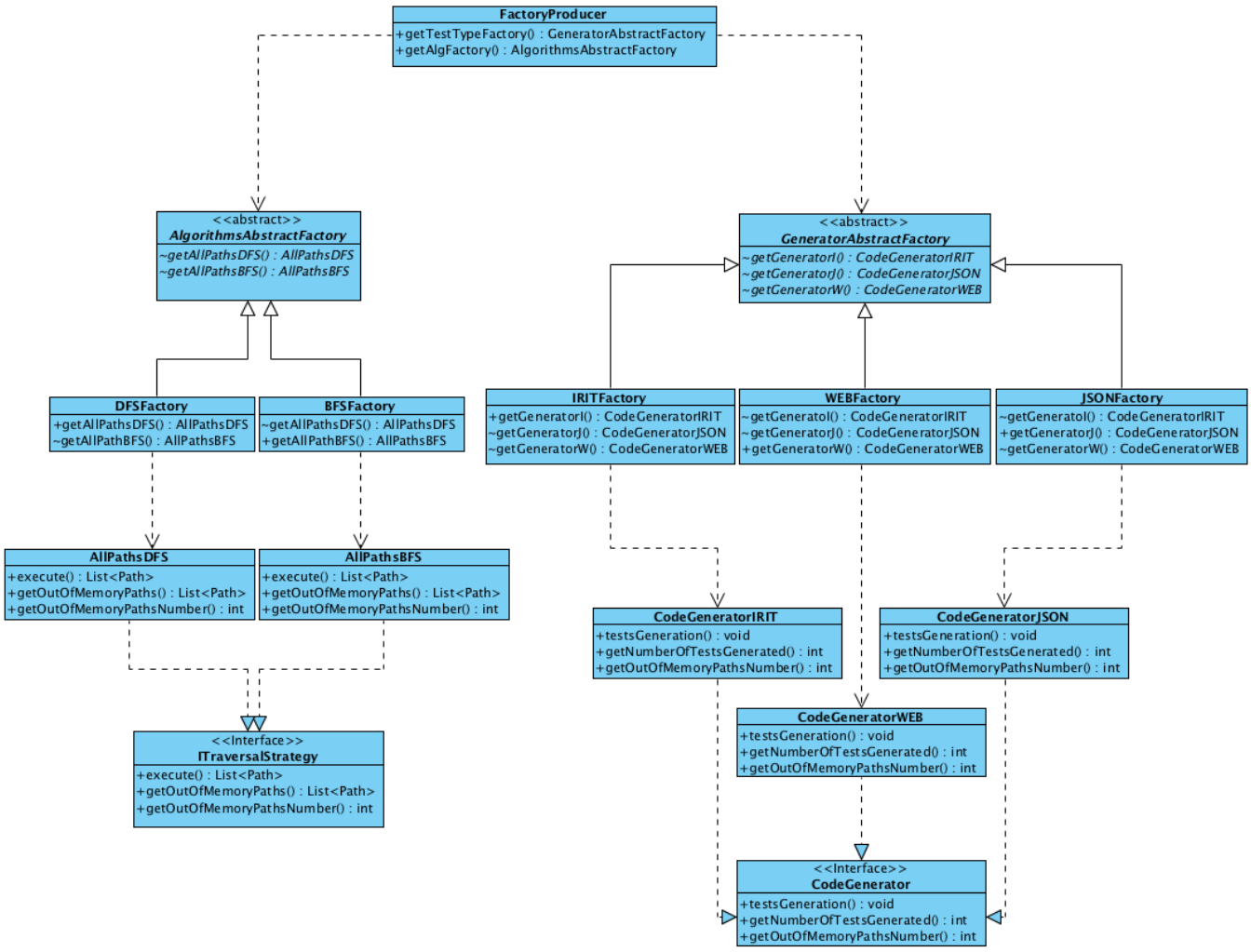


Figure 9.: Abstract Factory pattern applied to the Algorithms and Generators

The application of this pattern in these two types of modules was due to the need of configuring the system for several types of algorithms and several types of generators, unifying the way they are used. After the application of this pattern the type of algorithm and generator to be used for the generation of test cases is left to the client, who sets up the generation with the desired parameters.

Applying this pattern allows adding new algorithms or generators to TOM Generator, but, it involves changing the AbstractFactory (AlgorithmsAbstractFactory or GeneratorAbstractFactory) and all subclasses that depend on it. That is, to add a new algorithm, for example, it is required to change the AlgorithmsAbstractFactory to add the method that returns the new algorithm, with this change, it will be necessary to modify all subclasses that depend on the AlgorithmsAbstractFactory.

Although this represents a cost, the standardisation of the process of adding this type of modules allows greater ease of maintenance and adaptability of the TOM Generator.

After applying both patterns to the TOM Generator, a unification of the process of use and selection of the different types of modules was achieved. With this, the main purpose of allowing at runtime level that the choice of the type of modules to use is in charge of the user has been achieved. Thus different clients can use different modules.

4.2 Web Services layer

This section will explain the architecture adopted and the technologies used to develop the TOM Generator Web Services layer. To easy development of the Web Services layer, it was decided to use a framework. After a small analysis the choice fell on the Spring Framework[3]. There are two orders of reason for this choice. On the one hand, it is a Java based framework. Since TOM has been developed in Java, this seems like the natural choice. On the other hand, due to earlier familiarity with the framework, which will help reduce development time.

The Spring Framework is an open source application framework that intends to make Java EE development easier. Unlike single-tier frameworks, Spring intends to help structure entire applications consistently and productively, pulling together the best of single-tier frameworks to create a coherent architecture. To avoid the complications inherent in the initial configuration of Spring, it was decided to use Spring Boot. Spring Boot is a pre-configured suite to reduce the boiler plate configuration, providing the shortest way to have a Spring web application up and running easily.

To access the data layer Spring Data was used simply because it is already integrated into Spring and therefore it is not necessary to incorporate another framework for this purpose. As a database engine, the choice fell on MySQL. MySQL is a powerful, open source database management system. The choice fell on this technology because it is very stable and has a large community that helps maintain, debug and upgrade it.

After choosing the technologies to be used, the communication protocol to be used by the clients to communicate with the TOM Generator was defined. The choice fell into *Representational State Transfer (REST)* an architecture style for designing network applications that uses HTTP to make calls between machines. The choice fell on this style of development due to its simplicity when compared with the alternatives, such as SOAP or RPC. As a format for the requests, the choice fell on JSON because it is a light format of data exchange, and because it is easy to be read and generated by the system.

3 https://spring.io/

In Figure 10 the abstract architecture of the TOM Generator can be observed, the documentation of the developed API is available online[4].

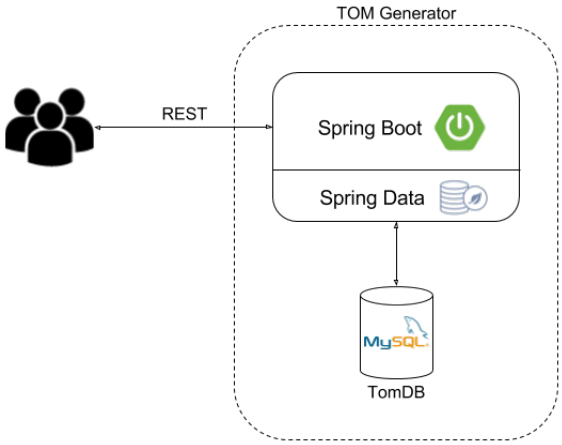


Figure 10.: Abstract architecture

4.2.1 Packages

An efficient organisation of the application code is essential for ease of maintenance. Then, during the creation of the Web Services layer, the organisation of the TOM Generator packages underwent several changes with the goal of improving it.

In Figure 11 the final organisation of the packages is presented. The next subsections will present the packages and their goals.

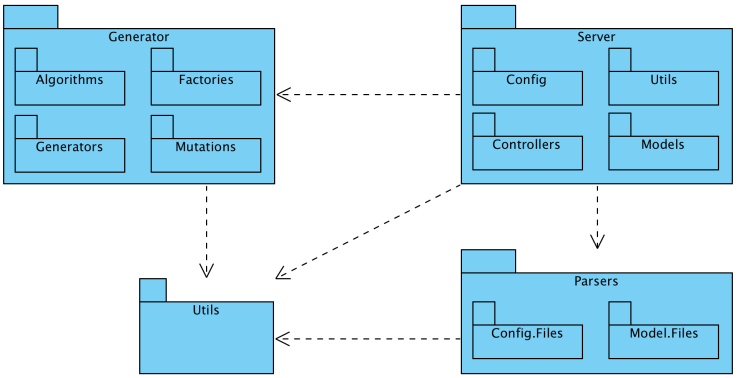


Figure 11.: Package organization

4 https://miguelpinto098.github.io/tom_generator/

Parsers Package

This package contains the code to read the contents of the files supported by the TOM Generator into its internal representation. The package is divided into two sub-packages, one that has the parsers for the configuration files (mappings, values and mutations), and another that has the parsers for the model files.

In Subsection 3.1.1 the contents of the files supported by the TOM Generator are explained, the utility of these files is also explained.

Generators Package

In the Generator Package, all the code responsible for generating the test cases is present. This package includes four sub-packages: algorithms, generators, mutations and factories.

Within the algorithms package is the code that allows obtaining the abstract test cases. The abstract test cases are obtained through graph traversal algorithms. At this moment, the TOM Generator has two such algorithms. One uses a Depth-First Search and the other uses Breadth-First Search. As previously mentioned, the TOM Generator is built for the possibility of adding new algorithms.

The Generators package contains also the code that converts abstract test cases into concrete test cases. Currently, TOM Generator is able to generate three types of test cases. These three types result from the improvements made to TOM Generator. In Section 3.2 each of these types was explained in more detail.

The Mutations package contains the code that allows the introduction of mutations in the test cases. TOM Generator allows the introduction of several types of mutations. However, these depend on the kind of test to be generated as explained in Section 3.2.

Finally, the Factories package allows access to all the features present in the previously mentioned packages. This package contains the factories that allow the instantiation of objects of those packages. Its goal is to make the system independent of how objects are created, represented or composed.

Server Package

The Server package contains all the classes that make the TOM Generator available as a service, that is, it contains all the code that is responsible for the WEB Services layer. The package contains also the classes that allow access to the data layer of the application.

The package is divided into four sub-packages: the Config, Controllers, Models, and Utils sub-packages. The Config sub-package contains the classes responsible for the following configurations:

- WEB Service security, such as creation of tokens for users and validation of tokens present in the requests
- Configuring Directories for storing user files
- Configuration of *Cross-origin resource sharing (CORS)*, a mechanism that allows restricted resources on a web page to be requested from another domain outside the domain from which the first resource was served

The Controllers sub-package contains all the controllers of the TOM Generator. The controllers control the execution of business tasks and coordinate the access to resources by users, that is, the controllers handle HTTP requests made by users to the TOM Generator. They are therefore primarily responsible for making the functionalities of TOM Generator available to users.

In the Models sub-package the classes that allow access to the data layer of the application can be found. This separation between controllers' logic and data access allows the business logic to be independent of the database, adding the ability to support multiple types of data storage.

Finally, the Utils sub-package contains the classes that provide access to users' files and to the messages used to answer user requests.

Utils Package

This package contains a set of utility classes used by the other packages. It was created to remove the repeated code in the different packages.

4.2.2 Database

Considering that the purpose is to provide TOM Generator as a service accessible to numerous clients, it has been decided to develop a database to persist users data. After analysing the TOM Generator, it was concluded that it was necessary to store the following information:

- User data;
- Generation requests;
- Generation results;
- User projects;

In Figure 12, the logical model of the developed database can be observed. In the database, the users of the framework are stored, as well as their projects. Considering the user projects, these are made up of multiple files. A project is made up of the model file and may contain mapping, mutation, and value files depending on the project (the contents of these files are shown

below). All information on generation requests made by users is stored. Namely, the project and the configurations used to perform the generation. The results obtained are also stored.

All files are stored in the file system of the server where the TOM Generator runs. In the database, only the paths to the files are stored.

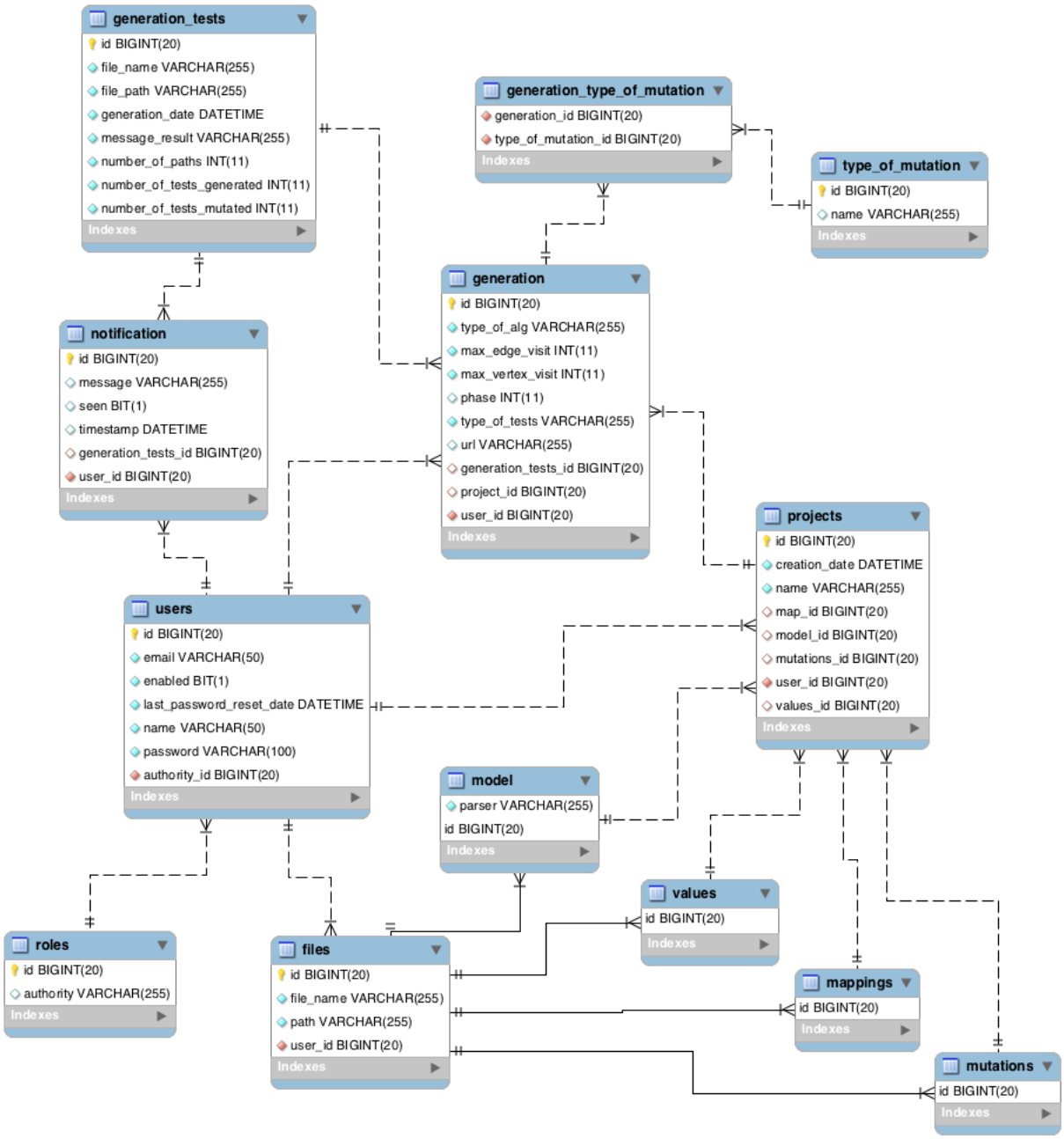


Figure 12.: Model of the database

4.2.3 Achieved Behaviour

Applied all the improvements and introduced the Web Services layer in the TOM Generator, the goal of making the component available as a service was reached. Figure 13 presents the architecture developed and implemented to meet the new needs.

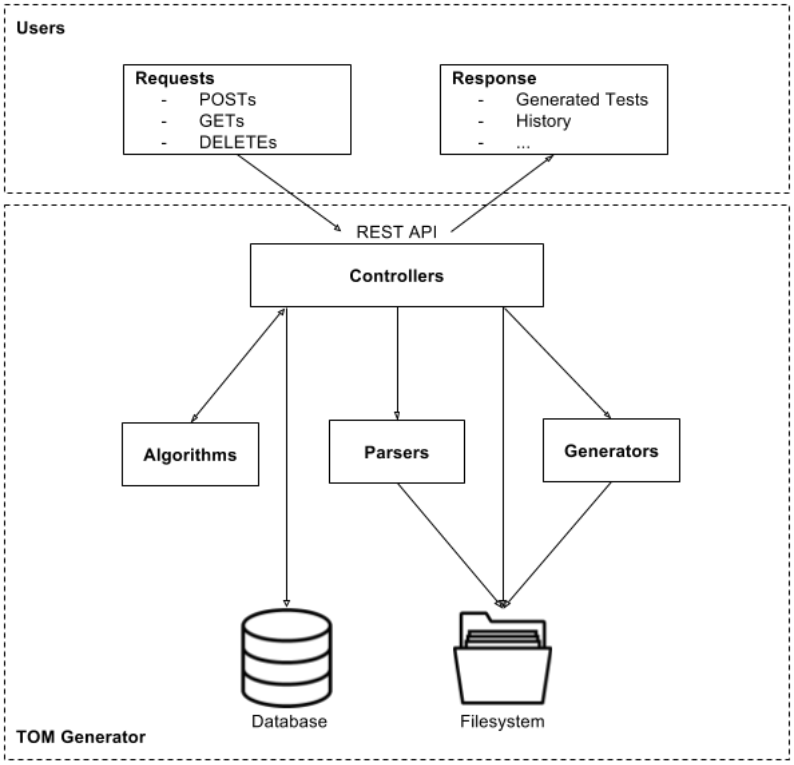


Figure 13.: Conceptual Architecture

After the inclusion of the new architecture in the TOM Generator, its mode of operation changed completely, becoming a service available to users via REST API. Users communicate with the TOM Generator via a REST API, all requests made through the developed API are handled by controllers who are responsible for providing users with access to all the features of the TOM Generator. After processing the requests, the controllers send their response to the users.

```
@RequestMapping(value = "/generation/history", method = RequestMethod.GET)
@Secured("ROLE_USER")
public String generationHistory(Principal principal) {

 JsonObject response = new JsonObject();
 User user = userRepo.findByEmail(principal.getName());
 List<GenerationRequest> requests = genRequestRepo.findAllByUser(user);

 if (requests.isEmpty()) {
  response.add(MESSAGE, gson.toJsonTree(ERR_EMPTYHISTORY));
  response.addProperty(SUCCESS, false);
 } else {
  List<GenerationRequest> result = new ArrayList<>();
  for (GenerationRequest request: requests) {
   if (request.getGenerationTests() != null){ result.add(request); }
  }

  if (result.isEmpty()) {
   response.add(MESSAGE, gson.toJsonTree(ERR_EMPTYHISTORY));
   response.addProperty(SUCCESS, false);
  } else {
   response.add(REQUESTS, gson.toJsonTree(result));
   response.addProperty(SUCCESS, true);
  }
 }
 return gson.toJson(response);
}
```

Listing 4.1: Example of a method from a controller

Listing 4.1 shows a method from a controller (HistoryController). The method presented is responsible for replying to all valid requests to the "/generation/history" route of the server. The requests addressed to this route ask for user's generation history. The method upon receiving the request will process it and sends the response in JSON format. Listing 4.2 presents an example of response.

```
{
   "requests":[
      {
         "id":"integer",
         "maxVertexVisit":"integer", "maxEdgeVisit":"integer",
         "algType":"string", "typeOfTests":"string",
         "testTypesToGenerate":[ { "id":"integer", "typeName":"string" } ],
         "project":{
            "id":"integer",
            "name":"string",
            "model":{ "id":"integer", "fileName":"string" },
            "values":{ "id":"integer", "fileName":"string" },
            "mutations":{ "id":"integer", "fileName":"string" },
            "mapping":{ "id":"integer", "fileName":"string" },
         },
         "user":{
            "id":"integer",
            "name":"string",
            "email":"string"
         },
         "generationTests":{
            "id":"integer",
            "fileName":"string",
            "numberOfPaths":"integer",
            "numberOfTestsWithMutations":"integer",
            "numberOfTestsGenerated":"integer",
            "generationDate":"string",
            "messageResult":"string"
         }
      }
   ],
   "success":"boolean"
}
```

Listing 4.2: Example of a JSON response

This is just an example of a TOM Generator controller method. The TOM Generator has seven controllers, responsible for responding to requests on 20 different routes. All requests to the server, except the authentication requests, must contain an access token in the request header, if they do not have the data are not provided.

4.3 TOM App

The creation of the Web Services layer supports the communication with TOM Generator through the developed API. While this is useful for tool integration, it does not support direct user interaction with the tool. TOM App was developed to facilitate the interaction of users with the API. In this section, the TOM App is presented.

TOM App is a web application that allows users to interact with all TOM Generator's functionality simply and interactively, thus avoiding the complexity inherent in making requests directly to the TOM Generator API.

4.3.1 Architecture

Conceptually the TOM App is an interface layer developed for the TOM Generator, with the intention of offering users an alternative to access its features. Consequently, users who want to use TOM Generator can choose to communicate directly with the API or use the TOM App to facilitate the process.

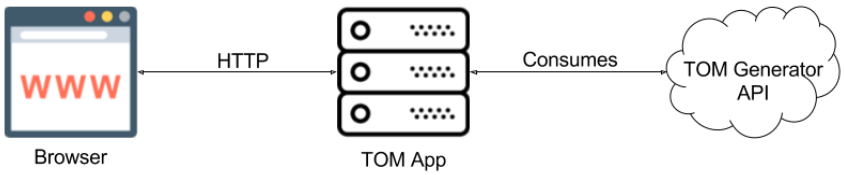


Figure 14.: Abstract Architecture

Figure 14 presents the abstract architecture of the TOM App. As can be seen, the users access the TOM App through their browsers, and the TOM App operates as an intermediary for communication with the TOM Generator. Although both applications run independently, the TOM App does not work without the TOM Generator, as its operation is coupled to the operation of the TOM Generator (the opposite is not true).

4.3.2 Mockups

Before starting the development of the TOM App, some low fidelity prototypes were created to support the design of its GUI. Using Moqups[5], a few mockups were made to give an approximate idea of the application's graphical user interface.

In the figure 15, two examples of the mockups created can be found.

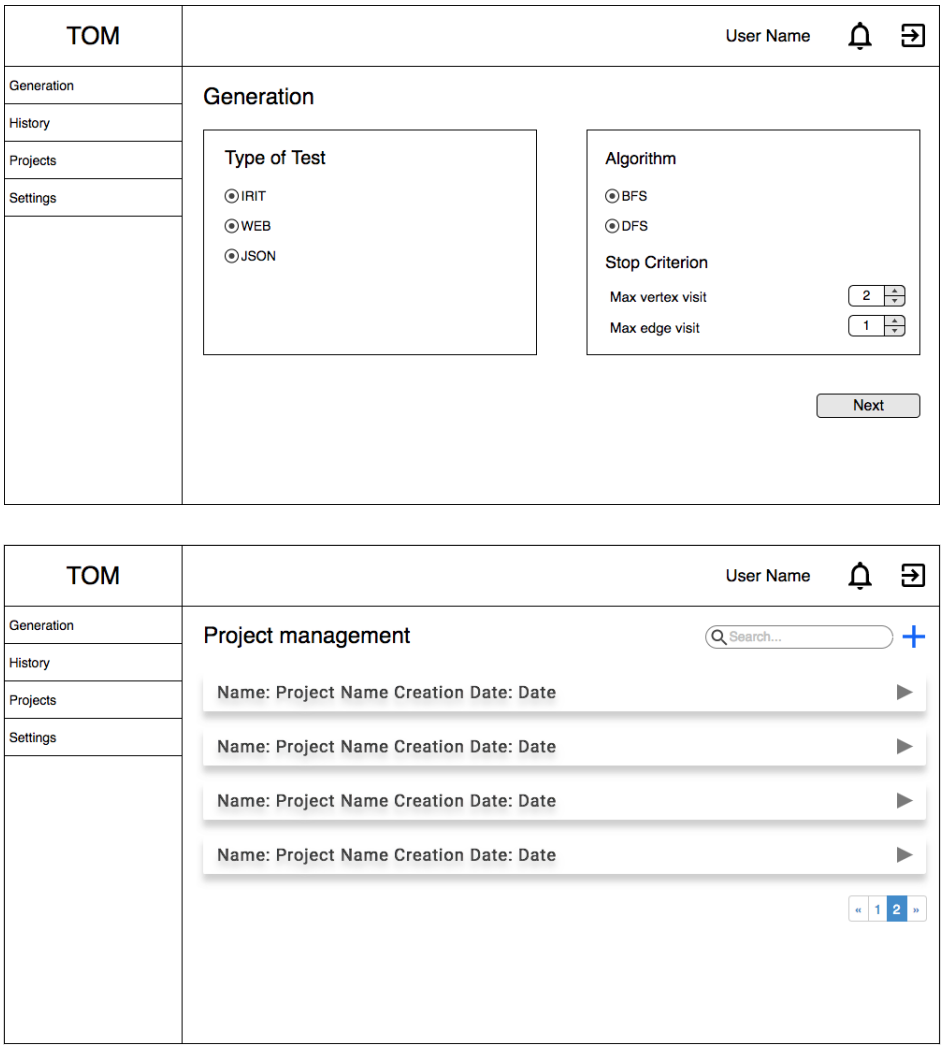


Figure 15.: TOM App Mockups Example

The mockups were presented to potential users so that some feedback could be collected. Following the mockups validation, the technologies to be used were chosen to start the development of the TOM App.

5 https://moqups.com

4.3.3 Technologies used

Since TOM App is a component of TOM Framework, the technologies used were chosen to facilitate the inclusion of new elements in the development of the framework. In this sense, the use of Angular 2[6], one of the latest technologies from Google, was decided, considering that this technology was already used to develop the TOM Editor.

Angular 2 is an open-source, front-end JavaScript framework for building client applications in HTML and either JavaScript or TypeScript. Angular 2 offers a components-based approach to web development.

The TOM App was developed using Angular Cli, the *Command Line Interface (CLI)* to automate the development workflow. It allows to easily create a application in Angular 2 using the best development practices. The list below shows some features offered by Angular CLI.

- Create an Angular application
- Run a development server with live reload support to preview the application during development
- Add features to the application
- Run application's unit tests
- Run application's end-to-end (E2E) tests
- Build the application for deployment

To use the Angular CLI, Node.js and NPM are required. Node.js[7] is used in the application base to run, test and build the application. Node.js helps in serving the Angular application from a "real" albeit light web server. NPM [8] is a package manager, it is used to access thousands of libraries that can be integrated into the application.

When creating the TOM App with Angular CLI, all source files and directories were created and the best-practices from the official Angular Style Guide applied. All NPM dependencies have been installed and TypeScript has been configured. The Karma[9] unit test runner and the Protractor[10] end-to-end test framework were also configured. Finally, environment files with default settings were created.

Figure 16 shows the initial skeleton of the Angular 2 application created. It is a simplified structure and it is only presented to explain some concepts inherent to the operation mode of

6 https://angular.io/
7 https://nodejs.org/en/
8 https://www.npmjs.com/
9 https://karma-runner.github.io/
10 http://www.protractortest.org/

Angular 2. The key idea behind an Angular 2 application is identifying components and composing them together as required to build an application. A component is an independent cohesive block of code which has the required logic, view, and data as one unit.

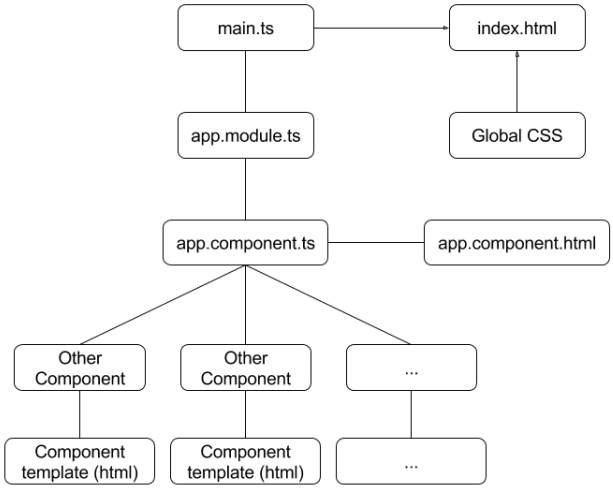


Figure 16.: General Architecture

The app is launched by bootstrapping the root module. Bootstrapping is an essential process as it is where the application is loaded and Angular 2 comes to life. The bootstrap process loads main.ts, which is the glue that combines the component and page together and the main entry point of the application. The app.module.ts file contains the AppModule that is the root module of our application. The module is configured to use AppComponent (the root component of our application, defined in the file app.component.ts) as the component to bootstrap, which will be rendered on the HTML element present in the index.html file. We use main.ts to import the AppModule component. The bootstrap process also starts the dependency injection system in Angular 2, however, this will not be explained in this dissertation.

Another core concept of any Angular 2 application is the component. The Angular team defines a component as "A component controls a patch of screen called a view., and declares reusable UI building blocks for an application"[11]. Attached to the component is its template (HTML) that will be rendered when the component is invoked. As seen in Figure 15, the whole application can be modelled as a tree of components, that is when the application is accessed the HTML page is rendered, and the component root is invoked together with all the visible components of the page at that moment. As the user interacts with the application, some components are removed, and others are added.

11 https://angular-2-training-book.rangle.io/

4.3.4 Functionalities

The TOM App provides access to all the features offered by TOM Generator, so the main features of the application are has follows:

- Project management
- Generation Requests
- Generation Results

In the following sections each of these features is explained in detail, and the final appearance of the application is presented.

Project management

Project management allows users to manage their projects. A project is a set of files referring to an interface to be tested. It must include a model of the interface and, optionally, the auxiliary files (mutations, values and mappings). Projects are a crucial component of the generation process since the projects contain the model files used to generate the test cases.

In the project management page, the user can consult, add and remove projects. In the specific case of the removal of projects, note that the test cases generated in this project will also be removed. In the Figures 17 and 18 the final aspect of the application for the project management functionality can be found.

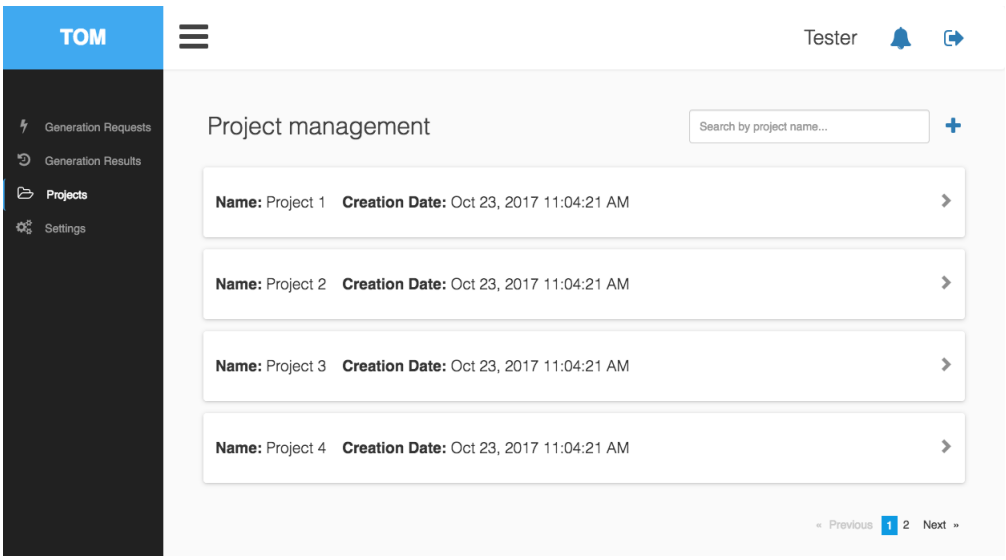


Figure 17.: Project management screen: Project Grouped

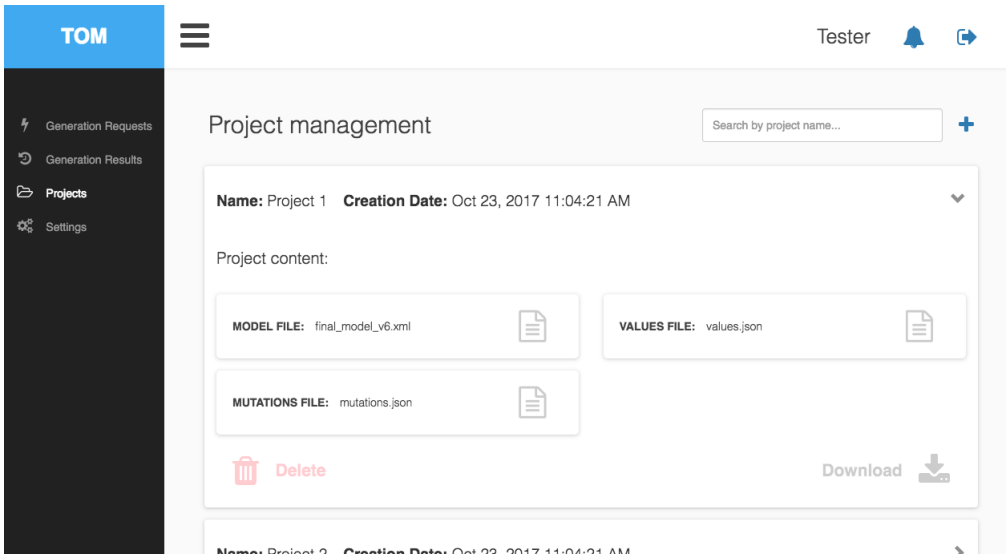


Figure 18.: Project management screen: Project Ungrouped

Generation Requests

The main functionality of the TOM App is to make generation requests of test cases to the TOM Generator. To make a generation request the user is guided through the different steps until his request is made. Figure 19 illustrates step 1.

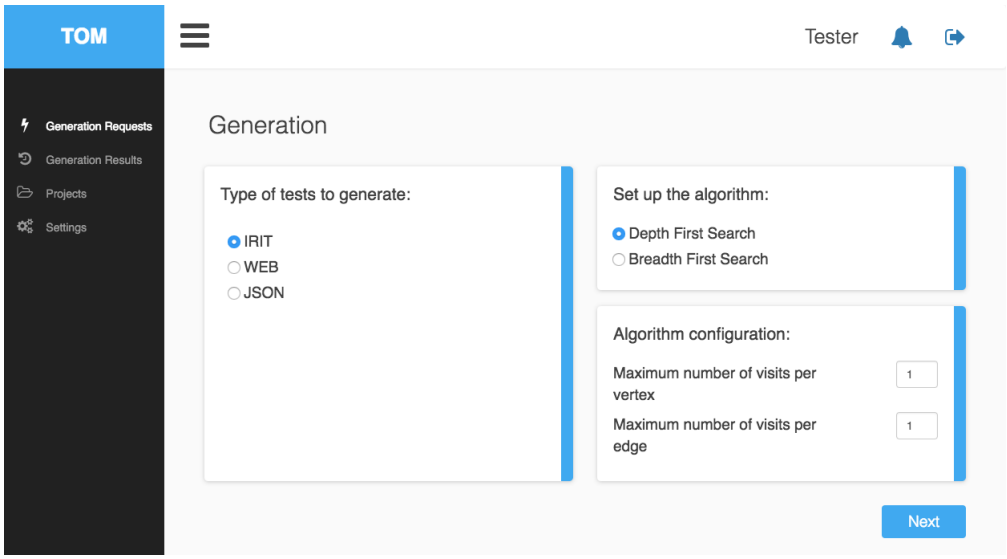


Figure 19.: Generation Requests: Step 1

In this step, the user can choose the type of tests to generate, the algorithm to be used and the configuration for the algorithm. After this step, the user is guided to the next step in which the user has to choose the project that he wants to use for the generation request. Figure 20 shows the final appearance of step 2.

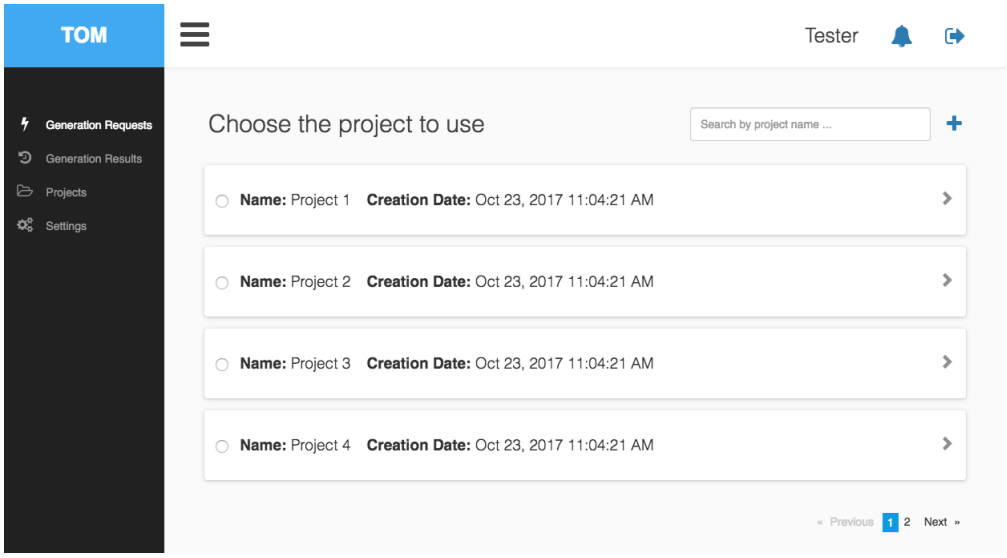


Figure 20.: Generation Requests: Step 2

After choosing the project, the user advances to the step 3, in which he has the possibility to choose the mutations that he wants to see introduced in the test cases. This step is optional and is only available for the types of tests that support mutations. In Figure 21 the final appearance of step 3 is shown.

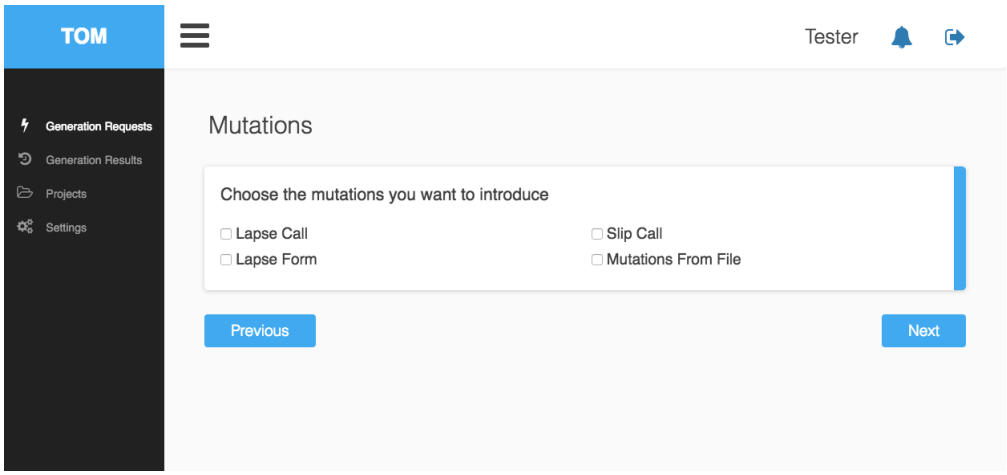


Figure 21.: Generation Requests: Step 3 (optional)

After this step, the user is presented with a review of his request so that he has an overview of the request and confirms his data. After confirming the request the user indicates to proceed with the generation (Figure 22).

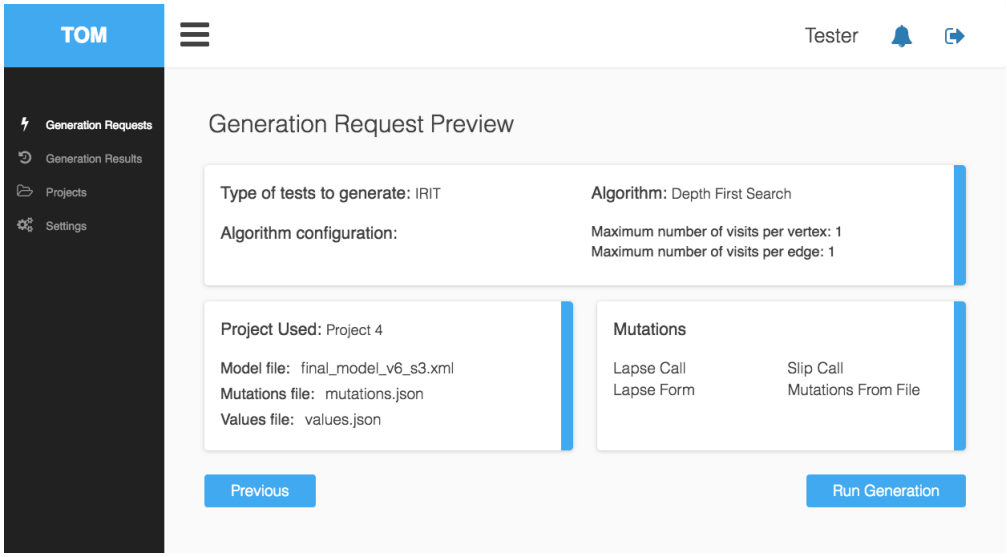


Figure 22.: Generation Requests: Step 4

When the generation request is confirmed, the request will be sent to the TOM Generator, and the TOM Generator will generate the test cases. At the end of the generation, the user receives a notification and can obtain the results of the generation on the page of the generations results.

Generations Results

The generations results page is where the user can consult the results of all their generation requests. For each generation request, a summary of the request and statistical data regarding the generation (for example number of generated tests) are presented.

In the Figures 23 and 24 the final aspect of this functionality is presented.

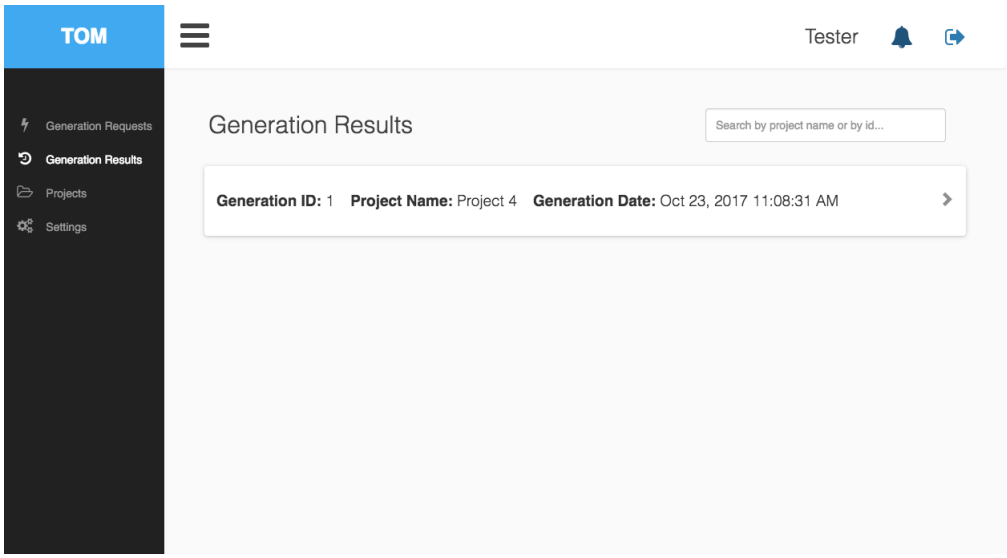


Figure 23.: Generation Results: Request Grouped

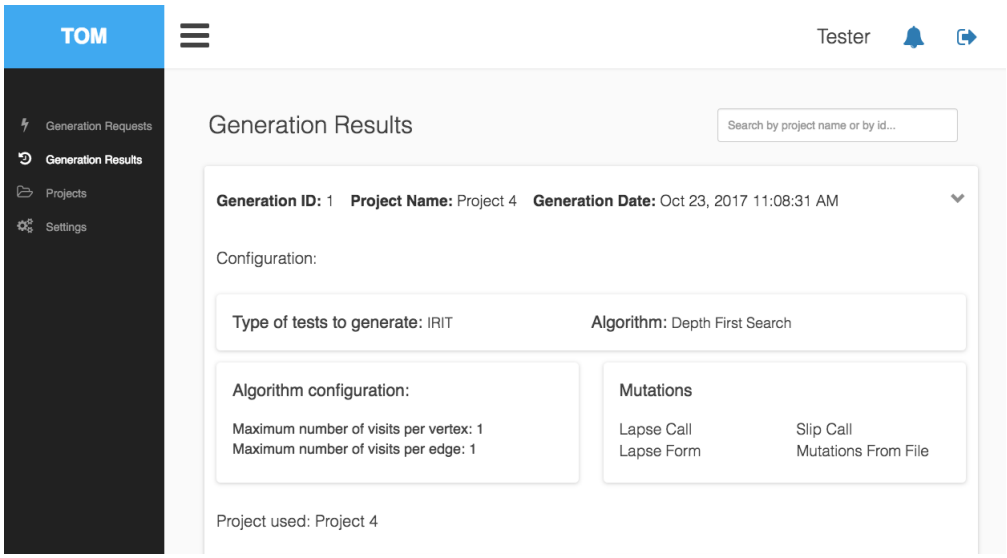


Figure 24.: Generation Results: Request Ungrouped

A generation result, in addition to presenting the request data and generation statistics data, also provides the possibility for the user to download the generated tests. After the download, the user can run the generated tests using the necessary tools for the process.

4.4 Summary

Throughout this chapter, all changes and improvements to the TOM Generator were described. Following these improvements, all changes made to the structure of this component were presented to introduce a Web Services layer in the TOM Generator. Finally, the developed architecture and the results of its implementation were presented, the design architecture allowed to maintain the existing modularity in the TOM Generator and thus preserve its adaptability.

The development of TOM App was also presented. The main stages of the application development process were explained and the reasons for some of the decisions taken were described. The final part of this chapter presents the main features of the application along with its final appearance. The main purpose of the application was achieved considering that all the features of the TOM Generator are accessible through the TOM App.

Chapter 5

Case Studies

In this chapter, we will demonstrate the many functionalities of the TOM Framework through several case studies. the goal is to confirm the proper functioning of the TOM Framework, using all the tools that constitute it to carry out the different case studies.

Three case studies will be presented to cover all the functionalities of the TOM Framework. The first case study is related to the generation of test scenarios based on task models, a case study proposed by IRIT. The second case study is related to a web application (OntoWorks), and the main objective is to compare the results obtained, after all the changes made throughout this dissertation, with the results achieved by Pinto (2017). Finally, in the third case study, the goal is to apply the entire TOM Framework test process to test a web application (in this case, the TOM App), thus obtaining an overview of the results of the whole process. In all the case studies presented, the results will be analysed and discussed.

5.1 Generating Scenarios from Task models

As mentioned previously, Pinto (2017) started the adaptation of the TOM Framework to work from task models, in this dissertation this adaptation was continued. The results obtained are presented in this case study.

The case study was proposed by IRIT and is based on task models for the *Flight Control Unit Software (FCUS)* system the Airbus A380 aircraft. First, details about the case study and its task modelling are provided, then the results obtained are described and discussed.

5.1.1 Flight Control Unit Software

As explained in Campos et al. (2017), the *Flight Control Unit (FCU)* is a hardware panel composed of several devices (such as buttons, knobs, displays, ...). It allows crew members to interact with

the Auto-Pilot and to configure flying and navigation displays. The FCUS is considered as a graphical interactive application for replacing the FCU hardware panel by graphical interfaces. It is composed of two interactive pages (displayed in Figure 25):

- EFIS_CP: Electronic Flight Information System Control Panel for configuring piloting and navigation displays
- AFS_CP: Auto Flight System Control Panel for the setting of the autopilot state and parameters

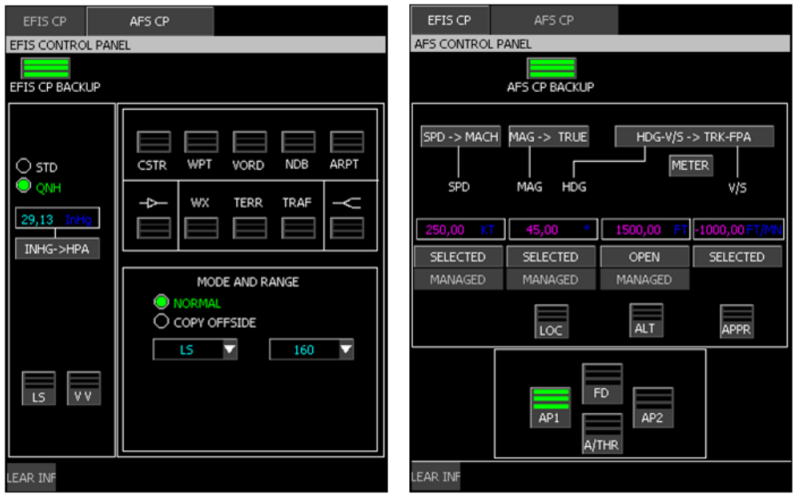


Figure 25.: The EFIS_CP (left side) and the AFS_CP (right side)

Using the FCUS interactive application the pilot can check the weather condition and verify if thunderstorms are on the flight route of the aircraft.

5.1.2 Task Modelling

The task used in this case study is "Check for thunderstorms and avoid them if necessary" and was provided by IRIT. This task can be carried out using the EFIS_CP to bring up the navigation display (ND) where weather radar information and the aircraft flight route are shown. If thunderstorms happen to be on the aircraft route, the pilots can avoid them using the AFS_CP, modifying the aircraft heading or its flight level. The task modelling for this goal is presented below.

The task "Check for thunderstorms and avoid them if necessary" is divided into two subtasks the "Check for thunderstorm" subtask and the "Avoid thunderstorm" subtask. Due to the size of the task models, only the "Check for thunderstorm" subtask is displayed (see Figure 26), the "Avoid thunderstorm" subtask is presented in Appendix A, along with the developed models.

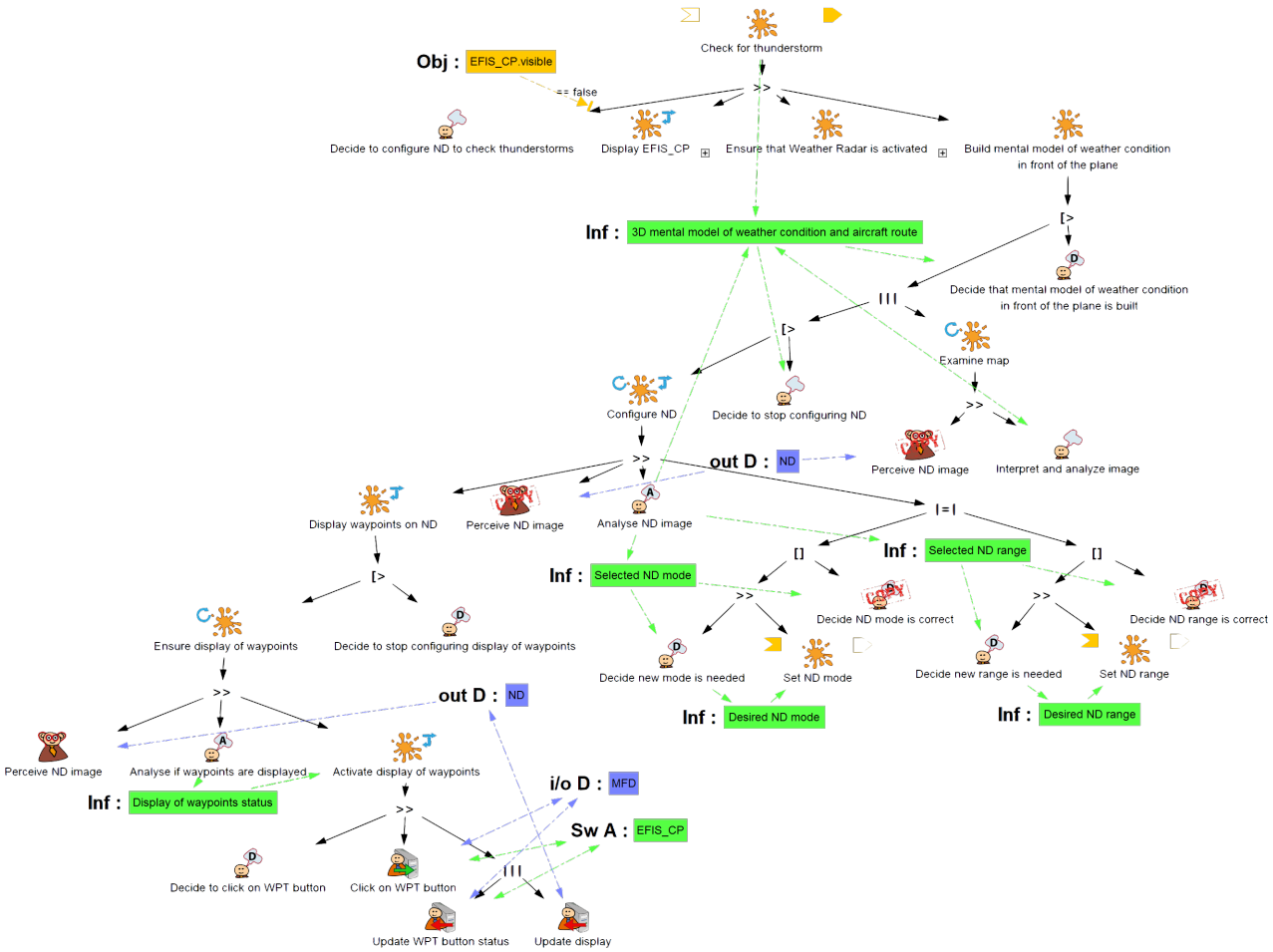


Figure 26.: Task model for the "Check for thunderstorm" task

According to the task model, to verify if a storm will appear on the aircraft's flight route, the pilot has first to decide to configure the navigation display ("Decide to configure ND to check for thunderstorms", cognitive user task). Then the pilot displays the EFIS_CP page if needed ("Display EFIS_CP", optional abstract task), and verifies that the weather radar is activated ("Ensure that weather radar is activated", abstract task). Finally, the pilot builds a mental model of the weather condition in front of the aircraft ("Build mental model of weather condition in front of the plane", abstract task). This last task consists of concurrently configuring the ND ("Configure ND", abstract task) and examining the map ("Examine map", abstract task). For configuring the ND, the pilot first displays the waypoints if needed ("Display waypoints on ND", optional abstract task). Then, after analysing the ND image, the pilot sets, in an order independent way, a new ND mode and a new ND range, if needed.

When the pilot decides that he has a valid image of the weather conditions ("Decide that mental model of weather condition in front of the plane is built", cognitive user task) ends the

"Check for thunderstorm" subtask. After the previous subtask, the pilot begins another subtask "Avoid thunderstorm" in which it decides if the plane's route is correct or should be changed. After the two subtasks are completed, the main task ("Check for thunderstorms and avoid them if necessary") ends.

5.1.3 Task model conversion

In order to generate the scenarios, it is necessary to convert the task model to a state machine (Model). For this, each state of the machine is defined as the set of possible tasks to execute at each moment of the interaction, the algorithm used for the conversion was described in more detail in Subsection 3.2.2 where the IRIT Scenarios are explained.

At this time the algorithm for generating state machines from task models was manually applied and the result double checked by the task model authors. However steps are being taken to make this process automatic

5.1.4 Scenarios Generation

Once the state machine was obtained, the test scenarios were generated. The main limitation of the TOM Generator is to assume that it is possible to check the model exhaustively. This limitation was revealed through this case study since the model generated from the full task model, besides being quite large, has many cycles (contained 92 states and 261 transitions). Generating scenarios from the developed model proved impracticable even allowing only one iteration in cycles, by setting up the algorithm to transverse every edge only once and each vertex at most twice. The *Java Virtual Machine (JVM)* run out of memory, due to exceeding the garbage collection overhead limit, after having found over 368 thousand paths, that is, more than 368 thousand test cases were generated, not counting the tests coming from mutations. Increasing the maximum memory allocation pool for the JVM to 4GB allowed the detection of more paths but the JVM still run out of memory before the process ended.

To overcome this limitation a set of strategies was defined to reduce the complexity of the task models while maintaining their relevant expressiveness and behaviour (Campos et al., 2017). The strategies reduce the complexity of the task models and consequently of the state machine. The strategies created can be classified in two abstract types of strategies:

- Modifying Operators on the Task Model - replacing operators that lead to complexity in the generated state machine (such as independence or concurrency operators).

- Removing Non-Interaction Tasks - simplify the task model by removing non-interaction tasks, as only interaction related tasks are relevant for the automated execution of test scenarios.

After applying the first type of strategies to the task model, the state machine obtained was reduced to 67 states and 98 transitions. With this state machine the generation process ended normally, with the maximum number of iterations restricted to one. The total number of paths detected was 27,840. If a maximum of two iterations was allowed, the total number of paths raised to 240,480. Applying the second set of strategies, the state machine obtained contained only 24 states and 51 transitions. With this simplified version, the number of test cases generated with the limitation of one iteration decreased to 1,584 and to 9,750 for the limitation of two iterations.

The simplifications introduced by the strategies reduce the set of behaviours that are possible, thus the number of test cases that are generated. However, the approach makes it possible to guarantee, that the analysis will focus on the relevant interactions with the SUT. The developed strategies and a more in-depth analysis of their implications can be found in Campos et al. (2017).

5.1.5 Conclusion

This case study allowed to deepen the integration of the TOM Generator with the tools under development in IRIT. Throughout this dissertation, optimizations in the TOM Generator were made to generate more complete scenarios, namely with the introduction of mutations and input data in the generated scenarios.

The studied strategies allow reducing the number of scenarios generated, through the simplification of the task model. These simplifications restrict the possible behaviours of the model to those deemed more relevant. Applying the strategies, it becomes possible to ensure full coverage of the simplified task model. The advantage is that it become clear, during the tasks manipulation process, which behaviours are being considered inside the analysis, and which behaviours are left out of the analysis.

Although the strategies defined are a good way of limiting the number of tests generated, these strategies can only be applied in the specific case of the task models, and can not be applied in other approaches. Thus, it is necessary to work on defining a coverage criterion for the state machines as an alternative to the current stopping criterion, so that the applied algorithms might generate the tests to reach a certain coverage rather than an exhaustive generation for all hypotheses.

5.2 Web Application: OntoWorks

The case study presented in this section is related to the OntoWorks web application, a SPARQL generic endpoint that allows the addition of ontologies and the execution of queries on them with the possibility to store SPARQL queries associated to the ontology. It is possible to check the list of available ontologies and access the list of queries related to each ontology. The user can register and authenticate on the platform. After performing the authentication, the user can load, edit and remove ontologies from the system, as well as associate and remove queries from ontologies.

This case study was originally performed by Pinto (2017). However, the results obtained in tests with mutations with the previous version of the TOM Generator were not very conclusive. Therefore, it was decided to carry out this case study again to make a comparison between the results obtained previously, and those obtained now after all the modifications made.

5.2.1 System Modelling

As this case study was previously performed and the application to be tested did not change since the last system modelling, the models constructed by Pinto (2017) were used.

The models used were constructed using the TOM Editor and can be consulted in Appendix B. The same models were used to make a comparison in similar circumstances, thus allowing us to observe with more clarify the evolution undergone by the framework along this dissertation.

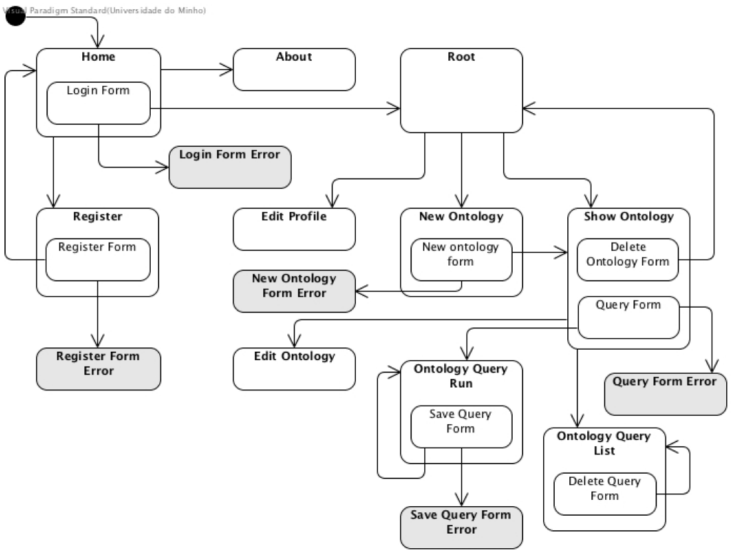


Figure 27.: Developed State Machine for OntoWorks (taken from Pinto (2017))

Figure 27 presents the representation of the developed state machine. In summary, the model has 15 states, 5 of which represent error states and are represented in the figure with grey shading. There are 24 transitions between states and 7 sub-states that correspond to the filling of forms in the web application. The validations added to the states are not shown in the figure.

5.2.2 Generation Request

The generation request for test cases was done through the TOM App. First, the OntoWorks project was imported into the Framework. The project consists of the four files present in Appendix B. After importing the project, the generation request was made using the same settings used by Pinto (2017):

- Type of tests to generate: WEB

  Browser to use: Chrome
- Algorithm: Breadth-First Search
- Algorithm Configuration:

  Maximum number of visits per vertex: 1

  Maximum number of visits per edge: 2
- All types of mutations were selected.

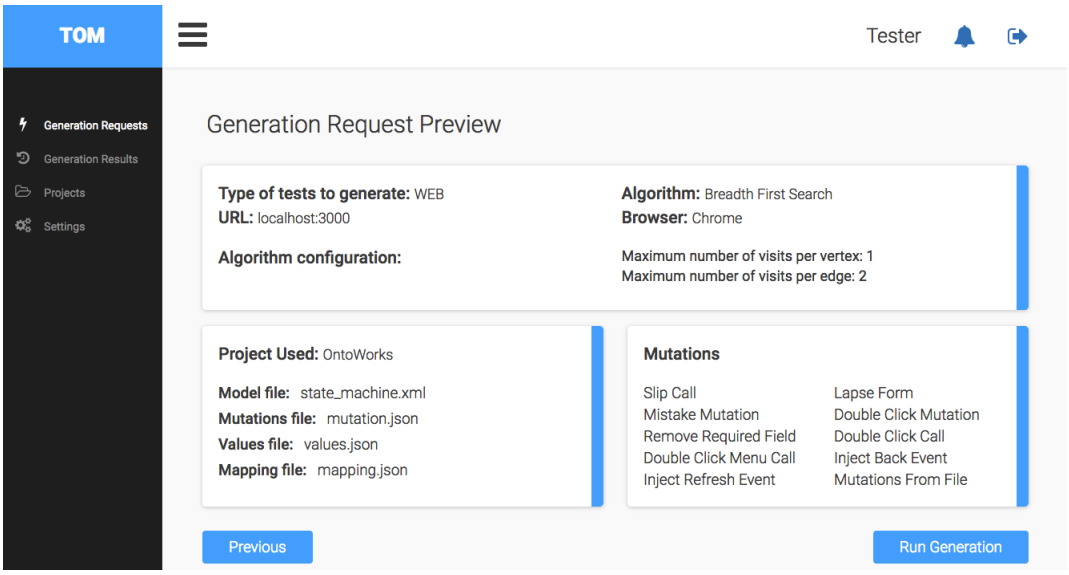


Figure 28.: OntoWorks: Generation Request Preview

Figure 28 presents the generation request preview. The generation request took some seconds to complete. The generation result can be seen in Figure 29. As can be seen, the TOM Generator took about 1015 milliseconds to generate the test cases. Twelve abstract paths were found, and from these paths, 114 tests were generated, of which 102 included mutations.

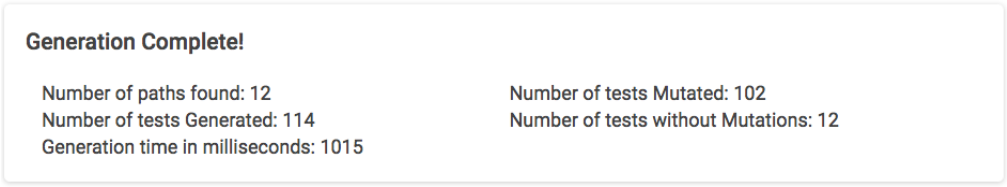


Figure 29.: OntoWorks: Generation Result

Comparing the results now obtained with those obtained previously it is concluded that there are some discrepancies in the number of tests generated. In Table 2 the results achieved in both generations can be observed.

	Old Results	New Results
Number of Paths found	12	12
Number of Tests Generated	3003	114
Number of Tests Mutated	2730	102
Files Generated	11	10

Table 2.: OntoWorks: Comparison of generation results

After analysing the table it it can be concluded that in both cases the number of paths found was the same (12 paths). However, there was a reduction of approximately 96% in the number of tests generated. This decrease is due to two factors, in the one hand, as previously mentioned, the generation process was changed so that an abstract path generates a single test instead of a group. In the other hand, the generation of repeated tests has been removed. Currently, TOM Generator no longer generates repeated tests, being these discarded. Precisely for this reason, the new version of the tool generated a test file less, since its contents were the same as the file contents with the tests without mutations. Despite the drastic reduction in the number of tests generated, the 114 tests generated ensure the same coverage of the 3003 tests previously generated, the main difference being the removal of the repeated tests and the fact that one test now generated corresponds to one group of tests of the previously generated ones.

5.2.3 Execution Analysis

After the generation request, the tests obtained were executed in OntoWorks. To run the generated tests, it was necessary to make the following configurations:

- create a maven project.
- add the tests to the project.
- add the necessary dependencies.
- configure the TestNG suite.

The previous configuration was the same as that used in the old generation. After performing the previous steps, the test suite is executed, and the results are obtained as the tests are executed. After running all the tests generated in OntoWorks, 46 failures were found in 114 tests. Figure 30 present the report with the final results of the test execution. In this figure, it is possible to realise that the total duration of the tests was superior to 1 hour and a half.

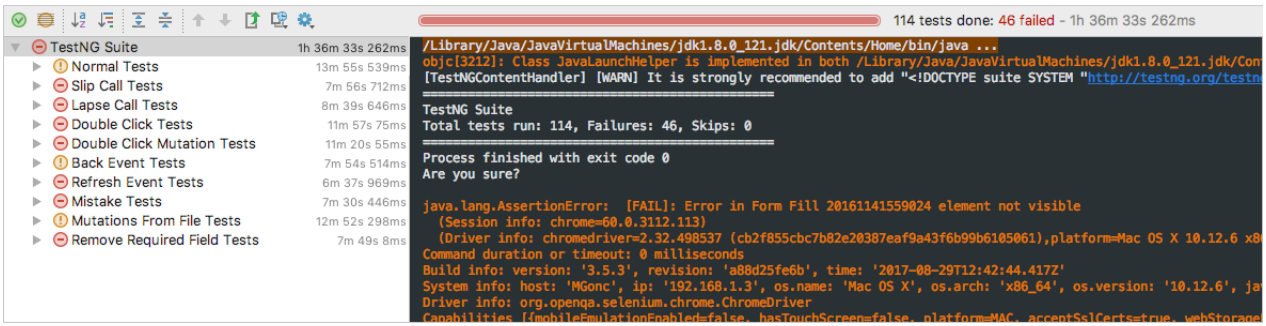


Figure 30.: OntoWorks: Execution Result

Observing the obtained results it is concluded that the relation between the number of failures obtained and the total number of tests is very similar to the previous results. However, with the fewest tests, the number of tests that need to be analysed is smaller. Nevertheless, the number of errors found was the same, thus, with less effort, the same problem was detected. In the Table 3 a comparison is made between the new obtained results and those obtained in the previous generation.

It can be seen from the table in non-mutated test cases (Normal), only one of the tests failed. After the analysis of the failed test, it is concluded that it is the same problem that was detected in the previous execution, and that the three tests that previously failed are all included in the test that failed in the new generation, due to all belonging to the same group. As concluded by Pinto (2017) this test fails because two input elements from different forms have the same value in their id attributes. Failure in this test (test2) occurs on all test types due to the same reason.

Test Type	Number of failed tests	
	Old Results	New Results
Normal	3	1
Lapse	168	6
Slip	67	5
Mistake	196	4
Double Click Submit	3	3
Remove Required Field	23	4
Double Click Call	15	7
Double Click Menu	6	-
Inject Back Event	261	12
Inject Refresh event	182	2
Mutations From File	11	2
Total	935	46

Table 3.: OntoWorks: Comparison of execution results

Files with tests where mutations were introduced have a higher rate of execution failures. Remember, these files are generated automatically by the tool. All mutations are introduced randomly in certain parts of the path, except mutations defined by the user in the mutation file, where they are entered as specified by the user. The results obtained in the test cases with mutations are explained below.

The tests where the "Lapse", "Slip", "Mistake", "Remove Required Field" and "Double Click Call" mutations were introduced have a relatively high number of failures. After an analysis of the tests and the obtained failures, it was concluded that the tests of these types fail for two reasons. The first and main reason is that the introduced mutations inhibit the appearance of some HTML element on the page, which is then unavailable at the time of running the tests. Listing 5.1 presents an excerpt of the code of a generated test, where a "Slip" mutation was introduced. In this test, the form to save a query was mutated, changing the order of execution of the required fields, that is, clicking the "Save Query" button (line 10), which was the first step, passed to last, and filling the description field (line 5), which was the last step, passed to first.

```
1 ...
2 try {
3     Reporter.log(" Mutation Slip <br>");
4     driver.findElement(By.id("desc")).sendKeys("...");
5     driver.findElement(By.id("desc")).getAttribute("value");
6     driver.findElement(By.id("name")).sendKeys("All classes");
7     driver.findElement(By.id("name")).getAttribute("value");
8     driver.findElement(By.id("query_start_save")).click();
9     Reporter.log("[Pass]: Form Filled save query <br>");
10    gonnaFail = 1;
11 } catch (WebDriverException _x) { ... }
12 ...
```

Listing 5.1: OntoWorks: Example of code generated for the slip mutation

With this mutation, the test fails because the "Description" field (line 6) is not visible without first clicking the "Save Query" button (line 10), so it is normal for the test to fail as the field will not be found.

The second reason is "false negatives", that is, when random mutations are introduced it is always assumed that the test should enter the case of error (variable "gonnaFail" set to '1'). However, this does not always happen. In Listing 5.2, we have an example of a test where a "Lapse" mutation was introduced.

```
1 ...
2 try {
3     Reporter.log(" Mutation Lapse <br>");
4     new Select(driver.findElement(By.id("prefixes_select"))).
          selectByVisibleText("PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-
          syntax-ns#>");
5     driver.findElement(By.id("prefixes_add")).click();
6     driver.findElement(By.id("prefixes_add")).click();
7     driver.findElement(By.id("prefixes_add")).click();
8     driver.findElement(By.id("prefixes_add")).click();
9     driver.findElement(By.id("prefixes_add")).click();
10    Reporter.log("[Pass]: Form Filled submit_query <br>");
11    gonnaFail = 1;
12 } catch (WebDriverException _x) { ... }
13 if ((gonnaFail == 1)) { ... }
14 ...
```

Listing 5.2: OntoWorks: Example of code generated for the lapse mutation

In this test, one of the steps in the form was removed, and it was assumed that this would lead the test to the error state ("gonnaFail = 1"). However, since the removed step was not required, the test failed because the error state of the form did not happen, that is, the test failed the validation of the error state. If the variable "gonnaFail" is set to 0, the test runs smoothly.

So, in tests that have mutations of type "Lapse", "Slip", "Mistake", "Remove Required Field" and "Double Clicks", the failed tests must be carefully analysed because they do not always represent errors in the page.

Regarding the tests where the "Inject Back Event" mutation was introduced, all tests failed. This kind of mutation introduced a go back event at the end of each step of the path. Since the back event changes the state to an earlier representation, the tests fail the validation of the new state.

The "Inject Refresh Event" mutation is introduced at the end of each step of the path. As the refresh of the page is entered between steps in the path, in the case of OntoWorks, the refresh does not influence the state, which is why the tests mostly pass except in two cases. One of the tests corresponds to a mutation of the the test previously mentioned (test2) and therefore it was not the target of analysis. After analysing the other test (test1) that failed, it was concluded that it should not have failed, and the failure occurred due to an error in the test. What happens is that the refresh was entered immediately after a click of a button, which caused the page to refresh without clicking the button to produce an effect. To solve this problem, a pause was introduced between the click of the button and the refresh, which made the test no longer fail. After solving this problem, the generation code was corrected so that these "false negatives" did not occur again. Another way to introduce this mutation would be to introduce the refresh event in the middle of completing a form, but this is not yet supported by the TOM Generator.

Regarding the tests where the mutations defined by the users were introduced, they should all pass because the user defines whether the test should pass or fail. However, two tests failed, one of them is a mutation of the test explained previously (test2), so it was not analysed. Analysing the other failed test, it was concluded that the generated test had an error. To reduce the number of generated test cases, all user-defined mutations that can be applied to a test are introduced simultaneously. In one of the forms, there were two mutations that could be applied, hence both were introduced. The problem arose because of the two mutations defined one causes an error state, the other does not. So, in the test, the variable "gonnaFail" should be set to '1'. However, it was set to '0', so the test failed, as the test will validate the normal state of the application and not the error state. After correction of the value of the variable, the test passed without problems. After the appearance of this error the TOM Generator was corrected, to introduce in the variable "gonnaFail" the stronger value of the introduced mutations.

5.2.4 Conclusion

In short, the results obtained were quite satisfactory. They were very similar to those obtained previously by Pinto (2017) but the improvements made to TOM Generator allowed to conduct a more accurate analysis with less effort of the obtained test.

The main conclusion obtained was the understanding of the reasons for the failures of the tests. After this case study, some improvements were introduced in the TOM Generator to correct some problems in the test cases generation, namely:

- Correction of tests generated with the "Inject Refresh Event" mutation.
- Correction of tests generated with user-defined mutations.

Test Type	Number of failed tests
Normal	1
Lapse	4
Slip	3
Mistake	4
Double Click Submit	1
Remove Required Field	4
Double Click Call	1
Inject Back Event	12
Inject Refresh event	1
Mutations From File	1
Total	32

Table 4.: OntoWorks: Final execution results

Despite the corrections made, the problem of "false negatives" has not been solved. After the corrections were made, another generation of test cases was performed, and all the tests that caused "false negatives" were manually corrected. The Table 4 presents the results of the execution of the tests after the changes.

As can be seen, with the removal of "false negatives" the results are even more encouraging. However, there are still tests that should not fail, but fail because the introduced mutations inhibit the appearance of some HTML element on the page, making the validations fail. However, this case study was beneficial to understand better the problems still existing in the tests generated.

5.3 Web Application: TOM App

This last case study, is intended to demonstrate the application of the TOM Framework to a web application. Based on the functionalities of the tools that make up the framework, it aims to create a system model using the TOM Editor, through the TOM App make the generation request to the TOM Generator and obtain the generation results. Finally, the results of execution of test cases will be analysed.

The web application chosen was the TOM App. As explained in Section 4.3, the TOM App is a web application that allows users to interact with all TOM Generator functionality simply and interactively, thus avoiding the complexity inherent in requesting directly to the TOM Generator API. Through this case study, in addition to obtaining an overview of the entire process of TOM Framework, the TOM App developed throughout this dissertation will also be tested.

5.3.1 System Modelling

The construction of the model was the first step taken to test the TOM App applying the framework. To develop the model one of the components of the framework, the TOM Editor, was used.

The TOM Editor has a series of features that support the development of a model in a semi-automatic way. Model building can be performed through the interaction capture mode, which captures user interaction with the page and builds states, transitions, and forms, or it can be manually created. However, the validations must be manually included in each state.

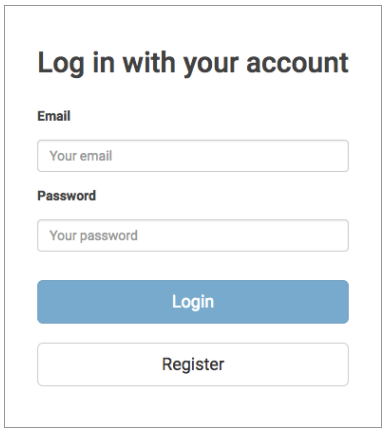


Figure 31.: Home Page

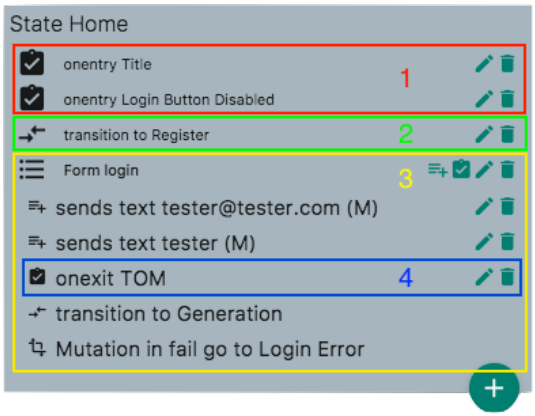


Figure 32.: Developed State

Figure 31 presents the application homepage, and Figure 32 shows the corresponding state developed through the TOM Editor. Both figures are examples used to explain the developed model.

The state, "Home", shown in Figure 32 includes a transition (green rectangle with number 2), two validations (red rectangle with number 1) and a form (yellow rectangle with number 3). The transition guides us to a new state, "Register", which permits us to register a new user. Concerning the two validations, one checks if the title of the page is correct, the other checks if at the moment the page is opened the "Login" button is disabled.

In the specific case of the form, it allows the entry of login data and gives access to the "Generation" state. Two mistake mutations have been added, one for the email field and another for the password. The introduced mutation will originate tests where the users enter invalid data in the email or password field, causing the error. With the introduced mutation, a transition to an error state ("Login Error") was added, for which the application transits in case of a login error. Finally, a form validation (blue rectangle with number 4) has been added to check if, after the form submission, the page changes and the label "TOM" appears on the next page.

Figure 33 presents the state machine that describes the developed model. The model consists of 21 states, 6 of which represent error states (shown in the figure with a grey shading). The navigation from these states is not represented because it is irrelevant to the evaluation. The model has 7 sub-states that correspond to the form filling in the web application and 24 state transitions. The 55 validations added to the states were not demonstrated in the figure, to make the interpretation easier.

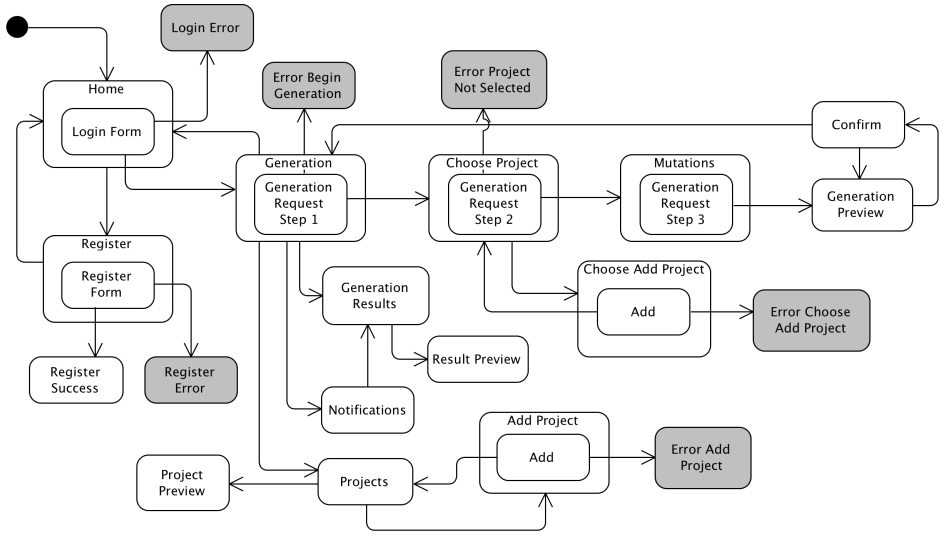


Figure 33.: Developed State Machine for TOM App

5.3.2 Generation Request

After completing the system model, the four project files (state machine, mapping, values and mutations, Appendix C) were exported and then imported into TOM App to perform the test generation request. The generation request was made using the following configuration:

- Type of tests to generate: WEB

  Browser to use: Chrome
- Algorithm: Depth-First Search
- Algorithm Configuration:

  Maximum number of visits per vertex: 2

  Maximum number of visits per edge: 1
- All types of mutations were selected.

Figure 34 presents the generation request preview. Once the generation request was made, generating the results took a few seconds.

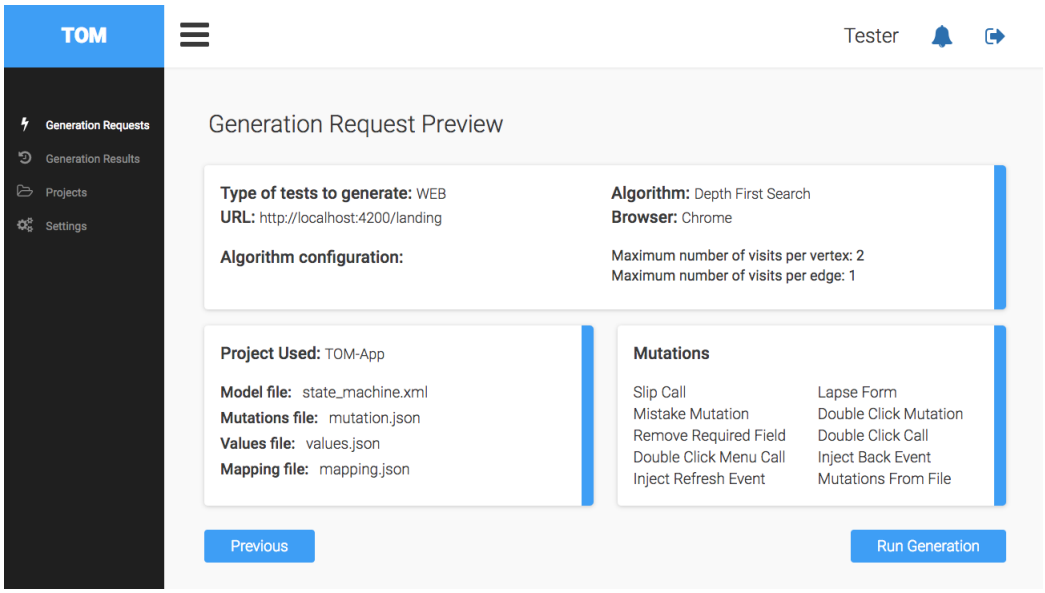


Figure 34.: TOM App: Generation Request Preview

The generation result can be seen in Figure 35. As can be seen, TOM Generator took about 2907 milliseconds to generate the test cases. 168 abstract paths were found, and from these paths, 1680 tests were generated, of which 1512 included mutations. Table 5 presents the summary of the generation result.

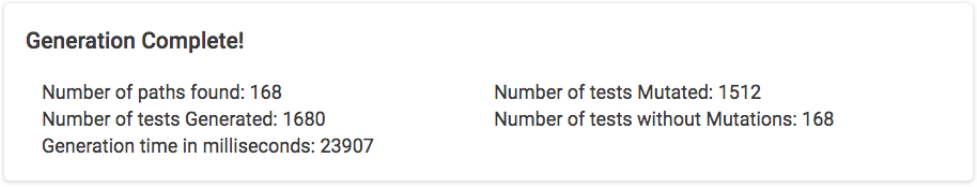


Figure 35.: TOM App: Generation Result

	Obtained Results
Number of Paths found	168
Number of Tests Generated	1680
Number of Tests Mutated	1512
Files Generated	10

Table 5.: TOM App: Generation results

5.3.3 Execution Analysis

After the tests generation, they were executed. The method used to run the tests was the same method used and explained in the Subsection 5.2.3. Testing took around 3 hours and a half to run, 951 failures were found in 1680 tests. Table 6 presents the results of the test execution.

Test Type	Number of failed tests
Normal	4
Lapse	96
Slip	131
Mistake	77
Double Click Submit	85
Remove Required Field	88
Double Click Call	134
Inject Back Event	168
Inject Refresh event	168
Mutations From File	0
Total	951

Table 6.: TOM App: Execution results

By doing a superficial analysis based only on the values obtained, the results are slightly alarming, since more than 50% of the tests failed. Therefore, all the failed tests were carefully analysed to discover the causes of the failures.

Analysing the tests without mutations ("Normal" type), in which 4 tests failed, all tests failed due to "Toasts Notifications". "Toasts Notifications" are messages that are displayed to the user for a few seconds. During the time they are presented, they can overlap HTML elements of the page, making them inaccessible. As the messages are temporary, duration of 1.5 seconds, these failures only occur when the load period of the page plus the time of the message is higher than the wait between each step (3 seconds). These failed tests happened sporadically in all the types of tests. While they do not represent a problem that will render the system unusable, they do highlight a situation that might be an annoyance to the user, due to the fact that he or she will have to wait for the notification to disappear before continuing. Assessing the severity of the problem could be done by an empirical evaluation process. To solve this problem, it was decided to change the position of the messages to a place where it does not overlap any HTML element of the page, just for the tests execution. This change allowed the "Toasts Notifications" to no longer cause the tests to fail. An alternative would be to increase the wait time between steps, but this would considerably increase the execution time of the tests.

For the tests where "Slip Call" mutations were randomly introduced, it was concluded that the vast majority of the failed tests (129 tests) were due to the assumption that the introduced mutation would cause an error state in the application ("false negatives"). However, because the error state does not occur, most tests fail in validations. Only 2 failed tests do not represent "false negatives". Both tests fail because some form field is not present on the page. Listing 5.3 is an excerpt from one of the failed tests.

```
1  ...
2  try {
3    Reporter.log(" Mutation Slip <br>");
4    driver.findElement(By.id("url")).sendKeys("www.google.pt");
5    driver.findElement(By.id("url")).getAttribute("value");
6    driver.findElement(By.id("DFS")).click();
7    driver.findElement(By.id("WEB")).click();
8    driver.findElement(By.id("1")).click();
9    driver.findElement(By.name("vertex")).sendKeys("1");
10   driver.findElement(By.name("vertex")).getAttribute("value");
11   driver.findElement(By.name("edge")).sendKeys("2");
12   driver.findElement(By.name("edge")).getAttribute("value");
13   gonnaFail = 1;
14 } catch (WebDriverException _x) { ... }
15 ...
```

Listing 5.3: TOM App: Example of code generated for the slip mutation

As can be seen, the first form field to be filled in is the 'URL' (line 5). The problem is that this field is only visible after choosing the 'WEB' option (line 8). This exchange in the filling order causes the test to fail, as the dependency between form fields is not met.

The tests where "Lapse", "Remove Required Field" and "Mistake" mutations were introduced, present a high number of failures. However, after analysing the failures, no error was found in the application, since all failed tests are mostly caused by the introduced mutation inhibiting the appearance of some HTML element. Only some failed tests represent "false negatives". This situation is reasonable since the mutations introduced in these tests, add errors to the form filling, so at the time of the form submission, as it has errors, the submit button is not visible causing the test to fail.

The tests where the "Double Clicks" ("Double Click Call" and "Double Click Submit") mutations were introduced, originated some suspicions. After a first analysis, it was concluded that the tests that failed were "false negatives". However, the fact that some tests execute with success, when assuming that they cause the error state, gave rise to some doubts, since none of the mutations introduced should originate a state of error. After, the analysis of these tests it was concluded that the tests of this type execute successfully, even assuming that an error state will occur ("gonnaFail $= 1$"), for two reasons. The first is that as the error state was not defined in the model (probably does not exist), it has no validations, so when the test checks the error state just ends, and as there were no errors, it executes successfully. The second reason is that after the inserted mutation at no time is the error state validated, that is, at no time is the value of variable "gonnaFail" checked. Thus, the test follows the ordinary course, running successfully, since the tests of this type should not cause the error state. After this analysis, the TOM Generator was corrected, so for this kind of mutation it is no longer assumed that the test will cause the error state.

The tests where the mutations "Inject Refresh Event" were introduced, all failed, which was a great surprise, since it was expected that all would pass. Firstly it was thought that they were "false negatives". However, after an analysis of the tests, it was verified that it was normal the tests to fail, since when a model window is open if a refresh is done the modal closes, which causes the test to fail. Then, as the application to test has numerous interactions that are made through "modals", it was considered *normal* the tests fail after the refresh when there are "modals" open. Note that in order to avoid this problem with the refresh button, the application would have to be redesigned.

Considering the tests where the introduced mutations were manually defined ("Mutations From File"), they all passed as expected, since in addition to defining the mutation to be introduced, its result is also described. Thus, only an error in the generation of the tests, or an error in the definition of the mutation files could provoke the opposite.

Finally, the tests where "Inject Back Event" mutations were introduced, all failed as they were supposed to.

5.3.4 Conclusion

After analysing the results, the "false negatives" were manually corrected, and all tests were rerun. Table 7 presents the results of the execution of the tests after the changes.

Test Type	Number of failed tests
Normal	0
Lapse	94
Slip	2
Mistake	56
Double Click Submit	0
Remove Required Field	88
Double Click Call	0
Inject Back Event	168
Inject Refresh event	168
Mutations From File	0
Total	576

Table 7.: TOM App: Final execution results

The realisation of this case study allowed to test the new component of the TOM Framework, the TOM App. Despite the large number of tests performed in the application, no error was found. Nevertheless, some pointers for future improvements were identified.

5.4 Summary

In this chapter, the application of the TOM Framework to three case studies was demonstrated. The case studies presented allowed an overview of the current state of the framework, as well as the correction of some errors in the TOM Generator. The results obtained are reasonable and meet the initial objectives.

Through the case studies, it was possible to conclude that the TOM Framework has the potential to assist in the detection of errors in GUIs. However, some improvements are needed, particularly in the generation of mutation tests, since a large number of "false negatives" are being generated.

Chapter 6

Conclusions

In this chapter, an analysis of the work developed during this dissertation is made. In the end, some ideas for improvements and new developments that may be part of the TOM Framework in the future are presented.

The TOM Framework provide a flexible and adaptable support for the application of MBT to GUIs. The TOM Framework is a set of tools designed to promote the introduction of advanced model-based testing technologies in the process of developing interactive software applications. At the beginning of this dissertation, the Tom Framework consisted of two components: the TOM Generator, a set of flexible and adaptable modules that aim to facilitate the application of MBT on GUIs, and the TOM Editor, a browser extension that supports the development of a model of a web application GUI.

6.1 Contributions

The main objectives of this dissertation were: optimise the test case generation process, improve the architecture of the TOM Framework, by developing the Web Services layer, and build a new component for the framework, the Web Application TOM App. All goals were met successfully.

Currently, the framework consists of three components: TOM Generator, TOM App and TOM Editor. The TOM Generator has been restructured so that its functionalities are accessible through a API, thus facilitating the integration of this component with other applications that wish to use its functionalities. This modification maintained the existing modular architecture in the component, and the possibility of adapting to new contexts. A new component, the TOM App, has been created to make it easier for users to interact with the API of the TOM Generator. Through TOM APP, the user is able to manage their projects, generate generation requests and observe generation results. The TOM Editor did not change in the course of this dissertation. The final state of the TOM Framework can be seen in Figure 36.

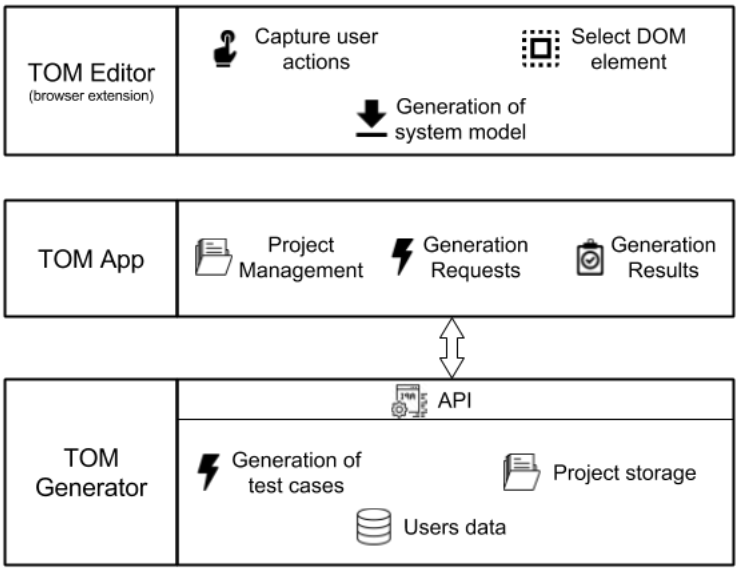


Figure 36.: TOM Framework

The application of the framework to the cases studies allowed to observe the results of the introduced improvements. The results obtained are quite encouraging, demonstrating that the framework is increasingly sophisticated, despite the limitations that still exist. Although there are still limitations in the generated tests, namely the "false negatives", it has been verified that the generated tests allow detecting problems in the implementation of user interfaces. As for the high number of tests generated, the TOM Generator has to be improved in order to ensure reasonable coverage of the model, since covering it fully is infeasible in certain situations.

In short, the results achieved in the development of this dissertation were:

- Improvement of scenarios generation.
- Improvement of test cases generation for WEB applications.
- Adaptation of the TOM Generator to the EmuCharts models.
- Adaptation of the TOM Generator to the generation of test cases in JSON.
- Construction of the TOM Generator database.
- Modification of the TOM Generator architecture, to make its functionality available through an API.
- Construction of an application to access the features of the TOM Generator, the TOM App.
- Application of the Framework to three case studies, analysing the results obtained.
- Contribution in the publication of two scientific articles (Campos et al., 2017; Pinto et al., 2017).

6.2 Future Work

During this dissertation, some limitations of the framework were identified, as well as a set of new functionalities to be implemented in the future. Following, a set of improvements/functionalities proposals to be implemented in future iterations of the framework is presented:

- Optimize the tests generation, namely solve the problem of "false negatives", since they are the principal cause of failed tests.
- Improve the TOM Editor component, to communicate with the TOM Generator API, allowing the user projects to be stored on the server.
- Development of a tool that allows the creation of the state machine through a task model automatically.
- Development of a component for the execution of test cases, where detailed and clear reports are obtained.
- Develop a coverage criterion for the model to limit the number of tests generated. When the model is too large, it is impossible to test all the possibilities, so it is crucial to implement a criterion to deal with these situations.
- Improve the generation of tests in JSON, particularly, the inclusion of mutations, or values for input fields.
- Adaptation of the TOM Generator to the possibility of testing multi-user applications.

Bibliography

ISO. ISO/IEC 9646-1. Information Technology — Open Systems Interconnection — Conformance Testing Methodology and Framework, Part 1: General Concepts. *International Standards Organisation. ISO/IEC, First Edition*, 1994.

Paul Ammann and Jeff Offutt. *"Introduction to Software Testing"*. Cambridge University Press, New York, NY, USA, 1 edition, 2008. ISBN 0521880386, 9780521880381.

Mark Blackburn, Robert Busser, and Aaron Nauman. Why model-based test automation is different and what you should know to get started. In *In International Conference on Practical Software Quality and Testing*, 2004. URL http://www.psqtconference.com/2004east/program.php.

Judy Bowen and Steve Reeves. UI-driven test-first development of interactive systems. In *Proceedings of the 3rd ACM SIGCHI Symposium on Engineering Interactive Computing Systems*, EICS '11, pages 165–174, New York, NY, USA, 2011. ACM. ISBN 978-1-4503-0670-6. doi: 10.1145/1996461.1996515. URL http://doi.acm.org/10.1145/1996461.1996515.

Judy Bowen and Steve Reeves. Ui-design driven model-based testing. *Innov. Syst. Softw. Eng.*, 9(3):201–215, September 2013. ISSN 1614-5046. doi: 10.1007/s11334-013-0199-6. URL http://dx.doi.org/10.1007/s11334-013-0199-6.

José C. Campos, Camille Fayollas, Célia Martinie, David Navarre, Philippe Palanque, and Miguel Pinto. Systematic automation of scenario-based testing of user interfaces. In *Proceedings of the 8th ACM SIGCHI Symposium on Engineering Interactive Computing Systems*, EICS '16, pages 138–148, New York, NY, USA, 2016. ACM. ISBN 978-1-4503-4322-0. doi: 10.1145/2933242.2948735. URL http://doi.acm.org/10.1145/2933242.2948735.

José Creissac Campos, Camille Fayollas, Marcelo Gonçalves, Célia Martinie, David Navarre, Philippe Palanque, and Miguel Pinto. A more intelligent test case generation approach through task models manipulation. *Proc. ACM Hum.-Comput. Interact.*, 1(1):9:1–9:20, June 2017. ISSN 2573-0142. doi: 10.1145/3095811. URL http://doi.acm.org/10.1145/3095811.

Rui Carvalho. A Comparative Study of GUI Testing Aproaches. Master's thesis, Faculty of Engineering of the University of Porto, 2016.

Paulo Jesus Cruz and J. Creissac Campos. Ambiente de geração, mutação e execução de casos de teste para aplicações Web. *Atas da Conferência Interação*, page 45–52, 2013. URL http://hdl.handle.net/1822/26583.

Peter Farrell-Vinay. *"Manage Software Testing"*. Auerbach Publications, Boston, MA, USA, 2007. ISBN 0849393833, 9780849393839.

Camille Fayollas, Célia Martinie, David Navarre, and Philippe Palanque. Engineering mixed-criticality interactive applications. In *Proceedings of the 8th ACM SIGCHI Symposium on Engineering Interactive Computing Systems*, EICS '16, pages 108–119, New York, NY, USA, 2016. ACM. ISBN 978-1-4503-4322-0. doi: 10.1145/2933242.2933258. URL http://doi.acm.org/10.1145/2933242.2933258.

Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns: Elements of Reusable Object-oriented Software*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1995. ISBN 0-201-63361-2.

ISTQB. *"Standard Glossary of Terms Used in Software Testing"*. International Software Testing Qualifications Board.

Paul C. Jorgensen. *"Software Testing: A Craftsman's Approach"*. Auerbach Publications, 4th edition, 2013.

V. Lelli, A. Blouin, B. Baudry, and F. Coulon. On Model-Based Testing Advanced GUIs. In *2015 IEEE Eighth International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*, pages 1–10, April 2015. doi: 10.1109/ICSTW.2015.7107403.

Célia Martinie, Philippe Palanque, and Marco Winckler. Structuring and composition mechanisms to address scalability issues in task models. In *Proceedings of the 13th IFIP TC 13 International Conference on Human-computer Interaction - Volume Part III*, INTERACT'11, pages 589–609, Berlin, Heidelberg, 2011. Springer-Verlag. ISBN 978-3-642-23764-5. URL http://dl.acm.org/citation.cfm?id=2042182.2042235.

Paolo Masci, Patrick Oladimeji, Yi Zhang, Paul Jones, Paul Curzon, and Harold Thimbleby. *PVSio-web 2.0: Joining PVS to HCI*, pages 470–478. Springer International Publishing, 2015. ISBN 978-3-319-21690-4. doi: 10.1007/978-3-319-21690-4_30. URL http://dx.doi.org/10.1007/978-3-319-21690-4_30.

Gioacchino Mauro, Harold Thimbleby, Andrea Domenici, and Cinzia Bernardeschi. Extending a user interface prototyping tool with automatic MISRA C code generation. In *Proceedings of the Third Workshop on Formal Integrated Development Environment, F-IDE@FM 2016, Limassol, Cyprus, November 8, 2016.*, pages 53–66, 2016. doi: 10.4204/EPTCS.240.4. URL https://doi.org/10.4204/EPTCS.240.4.

Atif Memon, Ishan Banerjee, and Adithya Nagarajan. GUI Ripping: Reverse Engineering of Graphical User Interfaces for Testing. In *Proceedings of the 10th Working Conference on Reverse Engineering*, WCRE '03, pages 260–, Washington, DC, USA, 2003. IEEE Computer Society. ISBN 0-7695-2027-8. URL http://dl.acm.org/citation.cfm?id=950792.951350.

Atif M. Memon. *A Comprehensive Framework For Testing Graphical User Interfaces*. PhD thesis, University of Pittsburgh, 2001.

Rodrigo M.L.M. Moreira and Ana C.R. Paiva. Pbgt tool: An integrated modeling and testing environment for pattern-based gui testing. In *Proceedings of the 29th ACM/IEEE International Conference on Automated Software Engineering*, ASE '14, pages 863–866, New York, NY, USA, 2014. ACM. ISBN 978-1-4503-3013-8. doi: 10.1145/2642937.2648618. URL http://doi.acm.org/10.1145/2642937.2648618.

Glenford J. Myers, Corey Sandler, and Tom Badgett. *The Art of Software Testing*. Wiley Publishing, 3rd edition, 2011. ISBN 1118031962, 9781118031964.

Bao N. Nguyen, Bryan Robbins, Ishan Banerjee, and Atif Memon. GUITAR: an innovative tool for automated testing of GUI-driven software. *Automated Software Engg.*, 21(1):65–105, March 2014. ISSN 0928-8910. doi: 10.1007/s10515-013-0128-9. URL http://dx.doi.org/10.1007/s10515-013-0128-9.

N. Nyman. Using Monkey Test Tools. *TQE - Software Testing and Quality Engineering Magazine*, 29(2):18–21, 2000.

Ana C. R. Paiva, João C. P. Faria, Nikolai Tillmann, and Raul A. M. Vidal. A model-to-implementation mapping tool for automated model-based gui testing. In *Proceedings of the 7th International Conference on Formal Methods and Software Engineering*, ICFEM'05, pages 450–464, Berlin, Heidelberg, 2005. Springer-Verlag. ISBN 3-540-29797-9, 978-3-540-29797-0. doi: 10.1007/11576280_31. URL http://dx.doi.org/10.1007/11576280_31.

Ana Cristina Ramada Paiva. *Automated Specification-Based Testing of Graphical User Interfaces*. PhD thesis, Faculty of Engineering of the University of Porto, 2006. URL https://web.fe.up.pt/~apaiva/PhD/PhDGUITesting.pdf.

Miguel Pinto. TOM Framework: Uma ferramenta de testes baseados em modelos para interfaces gráficas web. Master's thesis, Minho University, 2017.

Miguel Pinto, Marcelo Goncalves, Paolo Masci, and José Creissac Campos. Tom: a model-based gui testing framework. In *14th International Conference on Formal Aspects of Component Software*. Springer, Springer, 2017.

Raphael Rodrigues. Testes Baseados em Modelos. Master's thesis, Minho University, 2015.

José L. Silva, José Creissac Campos, and Ana C. R. Paiva. Model-based User Interface Testing With Spec Explorer and ConcurTaskTrees. *Electron. Notes Theor. Comput. Sci.*, 208:77–93, April 2008. ISSN 1571-0661. doi: 10.1016/j.entcs.2008.03.108. URL http://dx.doi.org/10.1016/j.entcs.2008.03.108.

Mark Utting, Alexander Pretschner, and Bruno Legeard. A taxonomy of model-based testing approaches. *Softw. Test. Verif. Reliab.*, 22(5):297–312, August 2012. ISSN 0960-0833. doi: 10.1002/stvr.456. URL http://dx.doi.org/10.1002/stvr.456.

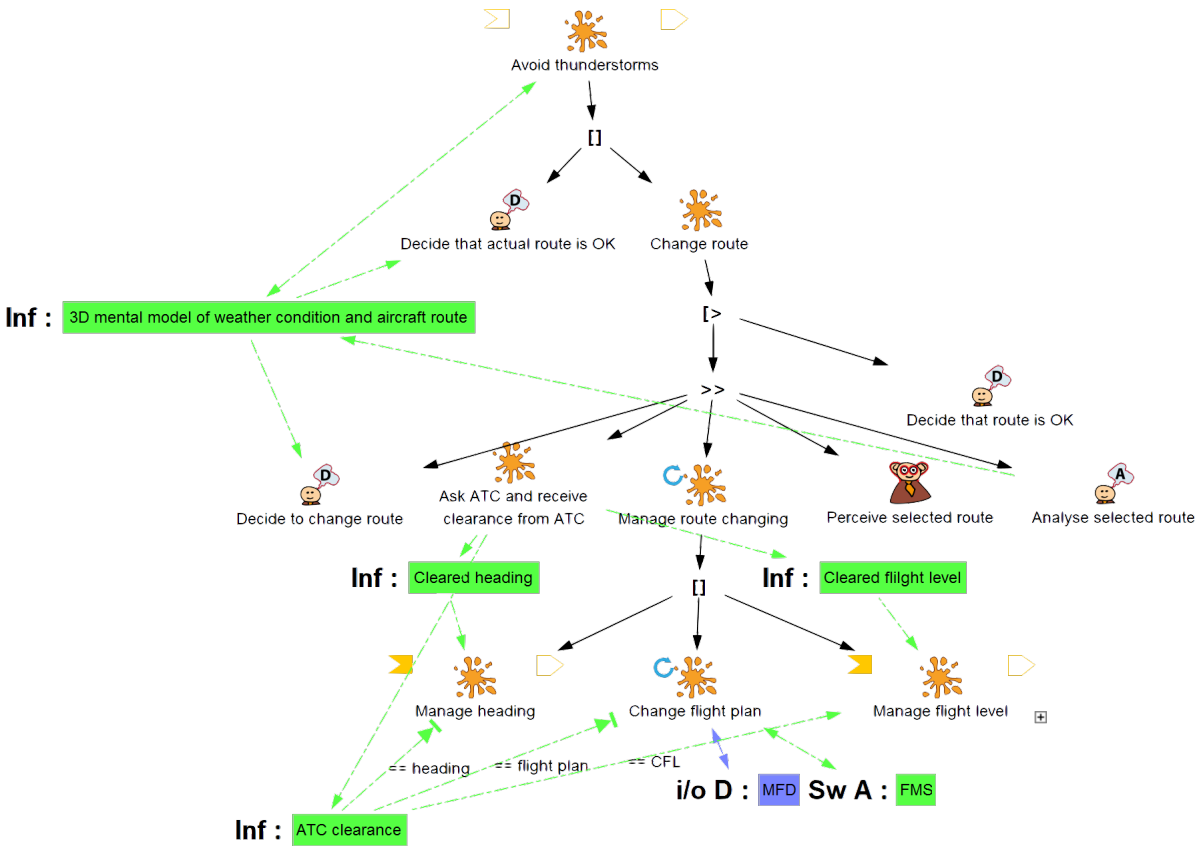
Xuebing Yang. *Graphic User Interface Modelling and Testing Automation*. PhD thesis, Victoria University, 2011.

X. Yuan, M. B. Cohen, and A. M. Memon. GUI interaction testing: Incorporating event context. *IEEE Transactions on Software Engineering*, 37(4):559–574, July 2011. ISSN 0098-5589. doi: 10.1109/TSE.2010.50.

Appendix A

IRIT Models

A.1 Sub Task - Avoid thunderstorm



A.2 State Machine

```
<?xml version="1.0" encoding="UTF-8"?>
<scxml xmlns="http://www.w3.org/2005/07/scxml" version="1.0" name="Model_v5"
    initial="State" final="State70">
  <state id="State72">
    <transition id="t198" target="State10" />
    <transition id="t191" target="State83" />
    <transition id="t18" target="State42" />
  </state>
  <state id="State71">
    <transition id="t194" target="State73" />
    <transition id="t197" target="State10" />
    <transition id="t18" target="State42" />
  </state>
  <state id="State92">
    <transition id="t216" target="State83" />
    <transition id="t217" target="State10" />
    <transition id="t18" target="State42" />
  </state>
  <state id="State28">
    <transition id="t217" target="State10" />
    <transition id="t215" target="State92" />
    <transition id="t18" target="State42" />
  </state>
  <state id="State90">
    <transition id="t217" target="State10" />
    <transition id="t214" target="State28" />
    <transition id="t18" target="State42" />
  </state>
  <state id="State83">
    <transition id="t205" target="State84" />
    <transition id="t217" target="State10" />
    <transition id="t18" target="State42" />
  </state>
  <state id="State89">
    <transition id="t217" target="State10" />
    <transition id="t211" target="State90" />
    <transition id="t18" target="State42" />
  </state>
  <state id="State88">
    <transition id="t217" target="State10" />
    <transition id="t212" target="State90" />
    <transition id="t18" target="State42" />
  </state>
  <state id="State87">
    <transition id="t212" target="State89" />
```

```
45      <transition id="t217" target="State10" />
46      <transition id="t18" target="State42" />
47      <transition id="t211" target="State88" />
48    </state>
49    <state id="State86">
50      <state id="State86.1" type="form">
51        <send label="ndModeCombobox" type="required" />
52        <transition type="form" id="t209">
53          <submit target="State87" />
54        </transition>
55      </state>
56      <transition id="t217" target="State10" />
57      <transition id="t18" target="State42" />
58    </state>
59    <state id="State85">
60      <transition id="t213" target="State86" />
61      <transition id="t217" target="State10" />
62      <transition id="t214" target="State28" />
63      <transition id="t18" target="State42" />
64    </state>
65    <state id="State84">
66      <transition id="t206" target="State85" />
67      <transition id="t217" target="State10" />
68      <transition id="t18" target="State42" />
69    </state>
70    <state id="State39">
71      <transition id="t315" target="State40" />
72      <transition id="t304" target="State72" />
73      <transition id="t18" target="State42" />
74    </state>
75    <state id="State38">
76      <transition id="t304" target="State72" />
77      <transition id="t318" target="State39" />
78      <transition id="t18" target="State42" />
79    </state>
80    <state id="State30">
81      <transition id="t317" target="State31" />
82      <transition id="t304" target="State72" />
83      <transition id="t18" target="State42" />
84    </state>
85    <state id="State40">
86      <transition id="t307" target="State30" />
87      <transition id="t304" target="State72" />
88      <transition id="t18" target="State42" />
89    </state>
90    <state id="State32">
91      <transition id="t314" target="State33" />
92      <transition id="t304" target="State72" />
```

```
93      <transition id="t318" target="State39" />
94      <transition id="t18" target="State42" />
95    </state>
96    <state id="State36">
97      <transition id="t311" target="State38" />
98      <transition id="t304" target="State72" />
99      <transition id="t18" target="State42" />
100   </state>
101   <state id="State35">
102     <transition id="t312" target="State38" />
103     <transition id="t304" target="State72" />
104     <transition id="t18" target="State42" />
105   </state>
106   <state id="State34">
107     <transition id="t311" target="State35" />
108     <transition id="t312" target="State36" />
109     <transition id="t304" target="State72" />
110     <transition id="t18" target="State42" />
111   </state>
112   <state id="State33">
113     <state id="State33.1" type="form">
114       <send label="ndRangeCombobox" type="required" />
115       <transition type="form" id="t313">
116         <submit target="State34" />
117       </transition>
118     </state>
119     <transition id="t304" target="State72" />
120     <transition id="t18" target="State42" />
121   </state>
122   <state id="State31">
123     <transition id="t316" target="State32" />
124     <transition id="t304" target="State72" />
125     <transition id="t18" target="State42" />
126   </state>
127   <state id="State81">
128     <transition id="t315" target="State82" />
129     <transition id="t304" target="State10" />
130     <transition id="t18" target="State42" />
131   </state>
132   <state id="State80">
133     <transition id="t304" target="State10" />
134     <transition id="t318" target="State81" />
135     <transition id="t18" target="State42" />
136   </state>
137   <state id="State73">
138     <transition id="t317" target="State74" />
139     <transition id="t304" target="State10" />
140     <transition id="t18" target="State42" />
```

```
141   </state>
142   <state id="State82">
143     <transition id="t307" target="State73" />
144     <transition id="t304" target="State10" />
145     <transition id="t18" target="State42" />
146   </state>
147   <state id="State75">
148     <transition id="t314" target="State76" />
149     <transition id="t304" target="State10" />
150     <transition id="t318" target="State81" />
151     <transition id="t18" target="State42" />
152   </state>
153   <state id="State79">
154     <transition id="t311" target="State80" />
155     <transition id="t304" target="State10" />
156     <transition id="t18" target="State42" />
157   </state>
158   <state id="State78">
159     <transition id="t312" target="State80" />
160     <transition id="t304" target="State10" />
161     <transition id="t18" target="State42" />
162   </state>
163   <state id="State77">
164     <transition id="t311" target="State78" />
165     <transition id="t304" target="State10" />
166     <transition id="t18" target="State42" />
167     <transition id="t312" target="State79" />
168   </state>
169   <state id="State76">
170     <state id="State76.1" type="form">
171       <send label="ndRangeCombobox" type="required" />
172       <transition type="form" id="t313">
173         <submit target="State77" />
174       </transition>
175     </state>
176     <transition id="t304" target="State10" />
177     <transition id="t18" target="State42" />
178   </state>
179   <state id="State74">
180     <transition id="t316" target="State75" />
181     <transition id="t304" target="State10" />
182     <transition id="t18" target="State42" />
183   </state>
184   <state id="State29">
185     <transition id="t216" target="State20" />
186     <transition id="t217" target="State71" />
187     <transition id="t18" target="State42" />
188   </state>
```

```
189   <state id="State91">
190     <transition id="t217" target="State71" />
191     <transition id="t215" target="State29" />
192     <transition id="t18" target="State42" />
193   </state>
194   <state id="State27">
195     <transition id="t217" target="State71" />
196     <transition id="t214" target="State91" />
197     <transition id="t18" target="State42" />
198   </state>
199   <state id="State20">
200     <transition id="t205" target="State21" />
201     <transition id="t217" target="State71" />
202     <transition id="t18" target="State42" />
203   </state>
204   <state id="State26">
205     <transition id="t217" target="State71" />
206     <transition id="t211" target="State27" />
207     <transition id="t18" target="State42" />
208   </state>
209   <state id="State25">
210     <transition id="t217" target="State71" />
211     <transition id="t212" target="State27" />
212     <transition id="t18" target="State42" />
213   </state>
214   <state id="State24">
215     <transition id="t211" target="State25" />
216     <transition id="t212" target="State26" />
217     <transition id="t217" target="State71" />
218     <transition id="t18" target="State42" />
219   </state>
220   <state id="State23">
221     <state id="State23.1" type="form">
222       <send label="ndModeCombobox" type="required" />
223       <transition type="form" id="t209">
224         <submit target="State24" />
225       </transition>
226     </state>
227     <transition id="t217" target="State71" />
228     <transition id="t18" target="State42" />
229   </state>
230   <state id="State22">
231     <transition id="t213" target="State23" />
232     <transition id="t217" target="State71" />
233     <transition id="t214" target="State91" />
234     <transition id="t18" target="State42" />
235   </state>
236   <state id="State21">
```

```
237     <transition id="t206" target="State22" />
238     <transition id="t217" target="State71" />
239     <transition id="t18" target="State42" />
240   </state>
241   <state id="State19">
242     <transition id="t191" target="State20" />
243     <transition id="t194" target="State30" />
244     <transition id="t198" target="State71" />
245     <transition id="t197" target="State72" />
246     <transition id="t18" target="State42" />
247   </state>
248   <state id="State66">
249     <transition id="t488" target="State67" />
250     <transition id="t481" target="State67" />
251   </state>
252   <state id="State61">
253     <transition id="t490" target="State62" />
254     <transition id="t86" target="State64" />
255     <transition id="t481" target="State67" />
256   </state>
257   <state id="State60">
258     <transition id="t489" target="State61" />
259     <transition id="t86" target="State64" />
260     <transition id="t481" target="State67" />
261   </state>
262   <state id="State55">
263     <transition id="t493" target="State56" />
264     <transition id="t429" target="State56" />
265   </state>
266   <state id="State50">
267     <transition id="t492" target="State51" />
268     <transition id="t434" target="State53" />
269     <transition id="t429" target="State56" />
270   </state>
271   <state id="State49">
272     <transition id="t491" target="State50" />
273     <transition id="t434" target="State53" />
274     <transition id="t429" target="State56" />
275   </state>
276   <state id="State41">
277     <transition id="t186" target="State10" />
278   </state>
279   <state id="State18">
280     <transition id="t190" target="State19" />
281     <transition id="t18" target="State42" />
282   </state>
283   <state id="State43">
284     <transition id="t106" target="State70" />
```

```
285     <transition id="t409" target="State70" />
286     <transition id="t408" target="State44" />
287   </state>
288   <state id="State65">
289     <transition id="t481" target="State67" />
290     <transition id="t478" target="State66" />
291   </state>
292   <state id="State63">
293     <transition id="t481" target="State67" />
294     <transition id="t468" target="State65" />
295   </state>
296   <state id="State64">
297     <transition id="t481" target="State67" />
298     <transition id="t467" target="State63" />
299   </state>
300   <state id="State62">
301     <state id="State62.1" type="form">
302       <send label="levelEditbox" type="required" />
303       <transition type="form" id="t463">
304         <submit target="State59" />
305       </transition>
306     </state>
307     <transition id="t481" target="State67" />
308     <transition id="t467" target="State63" />
309   </state>
310   <state id="State59">
311     <transition id="t481" target="State67" />
312     <transition id="t465" target="State60" />
313     <transition id="t86" target="State64" />
314   </state>
315   <state id="State58">
316     <transition id="t481" target="State67" />
317     <state id="State58.1" type="form">
318       <send label="levelEditbox" type="required" />
319       <transition type="form" id="t463">
320         <submit target="State59" />
321       </transition>
322     </state>
323     <transition id="t86" target="State64" />
324   </state>
325   <state id="State54">
326     <transition id="t429" target="State56" />
327     <transition id="t477" target="State55" />
328   </state>
329   <state id="State52">
330     <transition id="t429" target="State56" />
331     <transition id="t448" target="State54" />
332   </state>
```

```
<state id="State53">
  <transition id="t429" target="State56" />
  <transition id="t444" target="State52" />
</state>
<state id="State51">
  <transition id="t429" target="State56" />
  <state id="State51.1" type="form">
    <send label="headingEditbox" type="required" />
    <transition type="form" id="t435">
      <submit target="State48" />
    </transition>
  </state>
  <transition id="t444" target="State52" />
</state>
<state id="State48">
  <transition id="t429" target="State56" />
  <transition id="t440" target="State49" />
  <transition id="t434" target="State53" />
</state>
<state id="State47">
  <transition id="t429" target="State56" />
  <state id="State47.1" type="form">
    <send label="headingEditbox" type="required" />
    <transition type="form" id="t435">
      <submit target="State48" />
    </transition>
  </state>
  <transition id="t434" target="State53" />
</state>
<state id="State67">
  <transition id="t485" target="State69" />
  <transition id="t461" target="State57" />
</state>
<state id="State57">
  <transition id="t481" target="State67" />
  <transition id="t462" target="State58" />
</state>
<state id="State68">
  <transition id="t410" target="State68" />
  <transition id="t485" target="State69" />
</state>
<state id="State70" />
<state id="State56">
  <transition id="t485" target="State69" />
  <transition id="t446" target="State46" />
</state>
<state id="State46">
  <transition id="t429" target="State56" />
```

```
381       <transition id="t447" target="State47" />
382     </state>
383     <state id="State69">
384       <transition id="t409" target="State70" />
385       <transition id="t486" target="State70" />
386     </state>
387     <state id="State45">
388       <transition id="t485" target="State69" />
389       <transition id="t446" target="State46" />
390       <transition id="t429" target="State56" />
391       <transition id="t409" target="State70" />
392       <transition id="t410" target="State68" />
393       <transition id="t461" target="State57" />
394       <transition id="t481" target="State67" />
395     </state>
396     <state id="State44">
397       <transition id="t409" target="State70" />
398       <transition id="t487" target="State45" />
399     </state>
400     <state id="State17">
401       <transition id="t189" target="State18" />
402       <transition id="t18" target="State42" />
403     </state>
404     <state id="State16">
405       <transition id="t12" target="State17" />
406       <transition id="t14" target="State17" />
407       <transition id="t18" target="State42" />
408     </state>
409     <state id="State15">
410       <transition id="t13" target="State17" />
411       <transition id="t14" target="State17" />
412       <transition id="t18" target="State42" />
413     </state>
414     <state id="State14">
415       <transition id="t12" target="State15" />
416       <transition id="t13" target="State16" />
417       <transition id="t14" target="State17" />
418       <transition id="t18" target="State42" />
419     </state>
420     <state id="State13">
421       <transition id="t9" target="State14" />
422       <transition id="t14" target="State17" />
423       <transition id="t18" target="State42" />
424     </state>
425     <state id="State12">
426       <transition id="t10" target="State13" />
427       <transition id="t14" target="State17" />
428       <transition id="t6" target="State11" />
```

```
429     <transition id="t18" target="State42" />
430   </state>
431   <state id="State11">
432     <transition id="t5" target="State12" />
433     <transition id="t14" target="State17" />
434     <transition id="t18" target="State42" />
435   </state>
436   <state id="State42">
437     <transition id="t104" target="State43" />
438     <transition id="t185" target="State41" />
439   </state>
440   <state id="State10">
441     <transition id="t18" target="State42" />
442     <transition id="t6" target="State11" />
443     <transition id="t189" target="State18" />
444     <transition id="t185" target="State41" />
445     <transition id="t104" target="State43" />
446   </state>
447   <state id="State9">
448     <transition id="t70" target="State3" />
449     <transition id="t90" target="State10" />
450   </state>
451   <state id="State8">
452     <transition id="t72" target="State3" />
453     <transition id="t90" target="State10" />
454   </state>
455   <state id="State7">
456     <transition id="t70" target="State8" />
457     <transition id="t72" target="State9" />
458     <transition id="t90" target="State10" />
459   </state>
460   <state id="State6">
461     <transition id="t68" target="State7" />
462     <transition id="t90" target="State10" />
463   </state>
464   <state id="State5">
465     <transition id="t76" target="State6" />
466     <transition id="t90" target="State10" />
467   </state>
468   <state id="State4">
469     <transition id="t182" target="State5" />
470     <transition id="t90" target="State10" />
471   </state>
472   <state id="State3">
473     <transition id="t181" target="State4" />
474     <transition id="t90" target="State10" />
475   </state>
476   <state id="State2">
```

```
477     <transition id="t175" target="State3" />
478   </state>
479   <state id="State1">
480     <transition id="t174" target="State2" />
481     <transition id="t181" target="State4" />
482     <transition id="t90" target="State10" />
483   </state>
484   <state id="State">
485     <transition id="t17" target="State1" />
486   </state>
487 </scxml>
```

A.3 Values File

```
1  {
2      "NDModeCombobox": [
3          "0",
4          "1",
5          "-1"
6      ],
7      "NDRangeCombobox": [
8          "0",
9          "1",
10         "-1"
11     ],
12     "levelEditbox": [
13         "0",
14         "1",
15         "-1"
16     ],
17     "headingEditbox": [
18         "0",
19         "1",
20         "-1"
21     ]
22 }
```

A.4 Mutations File

```
[
   {
      "id" : "1",
      "type" : "lapse_call",
      "model_element": "t38"
   },
   {
      "id" : "2",
      "type" : "lapse_call",
      "model_element": "t60"
   },
   {
      "id" : "3",
      "type" : "lapse_form",
      "model_element": "t43"
   },
   {
      "id" : "4",
      "type" : "slip_call",
      "model_element": "t61",
      "target" : "t43"
   }
]
```

Appendix B

OntoWorks System Models

B.1 State Machine

```
<?xml version="1.0" encoding="UTF-8"?>
<scxml xmlns="http://www.w3.org/2005/07/scxml" version="1.0" name="Editor"
    initial="20161141559010">
  <state id="20161141559010">
    <onentry id="2017031127001" type="displayed?" />
    <onentry id="2017031127002" type="css" />
    <onentry id="2017031127003" type="contains" />
    <transition id="20161141559012" target="20161141559011" />
    <transition id="20161141559017" target="20161141559016" />
    <state id="sign in" type="form">
      <send label="s20161141559032" type="required" element="checkbox" />
      <send label="s20161141559033" type="required" />
      <send label="s20161141559034" type="required" />
      <send label="s20161141559035" type="required" element="checkbox" />
      <transition type="form" label="20161141559037">
        <submit target="20161141559038" />
        <error target="20161121851003" />
      </transition>
      <onexit id="2016114208001" type="contains" />
    </state>
  </state>
  <state id="20161141559011">
    <onentry id="20161141559013" type="default" />
    <onentry id="20161141559014" type="displayed?" />
    <onentry id="20161141559015" type="url" />
  </state>
  <state id="20161141559016">
    <onentry id="20161141559026" type="displayed?" />
    <onentry id="20161141559027" type="css" />
    <state id="new_user" type="form">
      <send label="s20161141559019" type="required" />
      <send label="s20161141559020" type="required" />
      <send label="s20161141559021" type="required" />
```

```
33      <send label="s20161141559022" type="required" />
34      <transition type="form" label="20161141559024">
35        <submit target="20161141559010" />
36        <error target="20161121851006" />
37      </transition>
38      <onexit id="20161141559028" type="contains" />
39    </state>
40  </state>
41  <state id="20161141559038">
42    <onentry id="2016114208002" type="displayed?" />
43    <onentry id="2016114208003" type="contains" />
44    <onentry id="2016114208004" type="default" />
45    <onentry id="2016114208007" type="default" />
46    <transition id="2016114208009" target="2016114208008" />
47    <transition id="20161121029002" target="20161121029001" />
48    <transition id="20161121029018" target="20161121029017" />
49  </state>
50  <state id="2016114208008">
51    <onentry id="2016114208019" type="attribute" />
52    <onentry id="2016114208020" type="enabled?" />
53    <onentry id="2016114208021" type="displayed?" />
54    <state id="new_ontology" type="form">
55      <send label="s2016114208010" type="required" />
56      <send label="s2016114208011" type="required" />
57      <send label="s2016114208012" type="required" element="checkbox" />
58      <send label="s2016114208014" type="required" />
59      <transition type="form" label="2016114208016">
60        <submit target="2016114208017" />
61        <error target="20161131049001" />
62      </transition>
63    </state>
64  </state>
65  <state id="2016114208017">
66    <onentry id="2016114208018" type="css" />
67    <onentry id="20161121029022" type="contains" />
68    <onentry id="20161121029023" type="default" />
69    <onexit id="20161121029024" type="enabled?" />
70  </state>
71  <state id="20161121029001">
72    <onentry id="20161121029025" type="displayed?" />
73    <onentry id="20161121029026" type="enabled?" />
74    <onentry id="20161121029027" type="displayed?" />
75    <onentry id="20161121029028" type="default" />
76    <onentry id="20161121029029" type="displayed?" />
77    <onentry id="20161121029030" type="contains" />
78    <onentry id="20161121029031" type="displayed?" />
79    <transition id="20161121029016" target="20161121029015" />
80    <transition id="20161121029019" target="20161121029011" />
```

```
81      <transition id="20161121029021" target="20161121029020" />
82      <state id="submit_query" type="form">
83        <send label="s20161121029003" type="required" element="selectbox" />
84        <send label="s20161121029004" type="required" element="checkbox" />
85        <send label="s20161121029005" type="required" element="checkbox" />
86        <send label="s20161121029006" type="required" element="checkbox" />
87        <send label="s20161121029007" type="required" />
88        <send label="s20161121029034" type="required" element="checkbox" />
89        <send label="s20161131049005" type="required" element="checkbox" />
90        <transition type="form" label="20161121029009">
91          <submit target="20161121029010" />
92          <error target="20161131049004" />
93        </transition>
94        <onexit id="2017031127004" type="enabled?" />
95      </state>
96      <state id="alert delete ontology" type="form">
97        <send label="s20161121425024" type="required" element="checkbox" />
98        <transition type="alert" label="20161121425023">
99          <submit target="20161141559038" />
100       </transition>
101       <onexit id="20161121425027" type="displayed?" />
102     </state>
103   </state>
104   <state id="20161121029010">
105     <onentry id="20161121029038" type="displayed?" />
106     <onentry id="20161121029039" type="not_displayed?" />
107     <onentry id="20161121029040" type="displayed?" />
108     <onentry id="20161121029041" type="contains" />
109     <onentry id="20161121029042" type="displayed?" />
110     <onentry id="20161121029043" type="enabled?" />
111     <onentry id="20161121029047" type="not_displayed?" />
112     <transition id="20161121029012" target="20161121029011" />
113     <state id="save query" type="form">
114       <send label="s20161121425003" type="required" element="checkbox" />
115       <send label="s20161121425004" type="required" />
116       <send label="s20161121425005" type="required" />
117       <transition type="ajax" label="20161121425002">
118         <submit target="20161121029010" />
119         <error target="20161131117001" />
120       </transition>
121       <onexit id="20161121425006" type="displayed?" />
122       <onexit id="2017031127005" type="not_displayed?" />
123     </state>
124   </state>
125   <state id="20161121029011">
126     <onentry id="20161121425013" type="contains" />
127     <onentry id="20161121425014" type="enabled?" />
128     <transition id="20161121029014" target="20161121029001" />
```

```
129     <state id="alert delete query" type="form">
130       <send label="s20161121425017" type="required" element="checkbox" />
131       <transition type="alert" label="20161121425016">
132         <submit target="20161121029011" />
133       </transition>
134       <onexit id="20161121425018" type="displayed?" />
135     </state>
136   </state>
137   <state id="20161121029015">
138     <onentry id="20161121425007" type="displayed?" />
139     <onentry id="20161121425008" type="default" />
140     <onentry id="20161121425009" type="enabled?" />
141     <onentry id="20161121425010" type="is_selected" />
142     <onentry id="20161121425011" type="enabled?" />
143     <onentry id="20161121425028" type="is_not_selected" />
144   </state>
145   <state id="20161121029017">
146     <onentry id="20161121425019" type="displayed?" />
147     <onentry id="20161121425020" type="attribute" />
148     <onentry id="20161121425021" type="enabled?" />
149   </state>
150   <state id="20161121029020">
151     <onentry id="20161121425025" type="displayed?" />
152     <onentry id="20161121425026" type="enabled?" />
153   </state>
154   <state id="20161121851003">
155     <onentry id="20161121851005" type="displayed?" />
156   </state>
157   <state id="20161121851006">
158     <onentry id="20161121851007" type="displayed?" />
159     <onentry id="20161121851008" type="contains" />
160   </state>
161   <state id="20161131049001">
162     <onentry id="20161131049002" type="displayed?" />
163     <onentry id="20161131049003" type="contains" />
164   </state>
165   <state id="20161131049004">
166     <onentry id="20161131049006" type="displayed?" />
167   </state>
168   <state id="20161131117001">
169     <onentry id="20161131117002" type="displayed?" />
170     <onentry id="20161131117003" type="contains" />
171   </state>
172 </scxml>
```

B.2 Values File

```
[
  { "2017031127002" : "rgba(0, 153, 218, 1)" },
  { "2017031127003" : "Login" },
  { "2016114208001" : "successfully" },
  { "s20161141559032" : "" },
  { "s20161141559033" : "admin@admin" },
  { "s20161141559034" : "1234567890" },
  { "s20161141559035" : "" },
  { "20161141559013" : "About project" },
  { "20161141559015" : "http://localhost:3000/about" },
  { "20161141559027" : "rgba(0, 153, 218, 1)" },
  { "20161141559028" : "A message with a confirmation link has been sent to
      your email address. Please follow the link to activate your account." },
  { "s20161141559019" : "12345" },
  { "s20161141559020" : "teste@teste" },
  { "s20161141559021" : "1234567890" },
  { "s20161141559022" : "1234567890" },
  { "2016114208003" : "My ontologies" },
  { "2016114208004" : "Public ontologies" },
  { "2016114208007" : "Welcome admin" },
  { "2016114208019" : "active" },
  { "s2016114208010" : "Periodic Table" },
  { "s2016114208011" : "Periodic table ontology" },
  { "s2016114208012" : "" },
  { "s2016114208014" : "/Users/mgonc/Desktop/[OLD] TOM/PeriodicTable.owl" },
  { "2016114208018" : "rgba(221, 255, 239, 1)" },
  { "20161121029022" : "was created" },
  { "20161121029023" : "admin" },
  { "20161121029028" : "Periodic Table" },
  { "20161121029030" : "Yes" },
  { "s20161121029003" : "PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-
      ns#>" },
  { "s20161121029004" : "" },
  { "s20161121029005" : "" },
  { "s20161121029006" : "" },
  { "s20161121029007" : "60" },
  { "s20161121029034" : "" },
  { "s20161131049005" : "" },
  { "s20161121425024" : "" },
  { "20161121029041" : "11" },
  { "s20161121425003" : "" },
  { "s20161121425004" : "All classes" },
  { "s20161121425005" : "Get all the classes of this ontology" },
  { "20161121425013" : "Table" },
  { "s20161121425017" : "" },
```

```
  { "20161121425008" : "Periodic Table" },
  { "20161121425020" : "admin" },
  { "20161121851008" : "error" },
  { "20161131049003" : "error" },
  { "20161131117003" : "Name" }
]
```

B.3 Mutations File

```
[
  {
    "type" : "lapse",
    "model_element": "s20161141559035",
    "fail" : "0"
  },
  {
    "type" : "slip",
    "model_element": "s20161141559019",
    "fail" : "0"
  },
  {
    "type" : "mistake",
    "model_element": "s20161141559022",
    "value" : "12345",
    "fail" : "1"
  },
  {
    "type" : "mistake",
    "model_element": "s2016114208011",
    "value" : "The periodic table ontology",
    "fail" : "0"
  },
  {
    "type" : "lapse",
    "model_element": "s2016114208012",
    "fail" : "0"
  },
  {
    "type" : "lapse",
    "model_element": "s2016114208014",
    "fail" : "1"
  },
```

```
  {
    "type" : "slip",
    "model_element": "s20161121029005",
    "fail" : "0"
  },
  {
    "type" : "slip",
    "model_element": "s20161131049005",
    "fail" : "0"
  },
  {
    "type" : "lapse",
    "model_element": "s20161121425004",
    "fail" : "1"
  },
  {
    "type" : "mistake",
    "model_element": "s20161121425005",
    "value" : "All the existent class for this ontology",
    "fail" : "0"
  }
]
```

B.4 Mapping File

```
{
  "2017031127001": {
    "how_to_find": "className",
    "what_to_find": "header-full-title",
    "what_to_do": "getText"
  },
  "2017031127002": {
    "how_to_find": "linkText",
    "what_to_find": "Home",
    "what_to_do": "getCssValue",
    "type_of_action": "background-color"
  },
  "2017031127003": {
    "how_to_find": "className",
    "what_to_find": "dropdown-toggle",
    "what_to_do": "getText"
  },
```

```
18   "20161141559012": {
19     "how_to_find": "linkText",
20     "what_to_find": "About us",
21     "what_to_do": "click"
22   },
23   "20161141559017": {
24     "how_to_find": "cssSelector",
25     "what_to_find": "#header-full-top > DIV:nth-child(1) > NAV:nth-child(2) >
           DIV:nth-child(1) > A:nth-child(1)",
26     "what_to_do": "click"
27   },
28   "2016114208001": {
29     "how_to_find": "cssSelector",
30     "what_to_find": "#sb-site > DIV:nth-child(1) > SECTION:nth-child(3) > DIV:
           nth-child(1) > DIV:nth-child(1) > DIV:nth-child(1) > DIV:nth-child(1)
           > DIV:nth-child(1) > DIV:nth-child(1)",
31     "what_to_do": "getText"
32   },
33   "s20161141559032": {
34     "how_to_find": "className",
35     "what_to_find": "dropdown-toggle",
36     "what_to_do": "click"
37   },
38   "s20161141559033": {
39     "how_to_find": "name",
40     "what_to_find": "user[email]",
41     "what_to_do": "sendKeys"
42   },
43   "s20161141559034": {
44     "how_to_find": "name",
45     "what_to_find": "user[password]",
46     "what_to_do": "sendKeys"
47   },
48   "s20161141559035": {
49     "how_to_find": "className",
50     "what_to_find": "translation_missing",
51     "what_to_do": "click"
52   },
53   "20161141559037": {
54     "how_to_find": "cssSelector",
55     "what_to_find": "#header-full-top > DIV:nth-child(1) > NAV:nth-child(2) >
           DIV:nth-child(2) > DIV:nth-child(2) > FORM:nth-child(1) > BUTTON:nth-
           child(6)",
56     "what_to_do": "submit"
57   },
58   "20161141559013": {
59     "how_to_find": "cssSelector",
```

```
60      "what_to_find": "#sb-site > DIV:nth-child(1) > SECTION:nth-child(3) > DIV:
            nth-child(1) > DIV:nth-child(1) > DIV:nth-child(1) > DIV:nth-child(1)
            > DIV:nth-child(1) > H2:nth-child(1)",
61      "what_to_do": "getText"
62    },
63    "20161141559014": {
64      "how_to_find": "cssSelector",
65      "what_to_find": "#sb-site > DIV:nth-child(1) > SECTION:nth-child(3) > DIV:
            nth-child(1) > DIV:nth-child(1) > DIV:nth-child(1) > DIV:nth-child(1)
            > DIV:nth-child(2)",
66      "what_to_do": "getText"
67    },
68    "20161141559015": {
69      "how_to_find": "id",
70      "what_to_find": "default",
71      "what_to_do": "getText"
72    },
73    "20161141559026": {
74      "how_to_find": "className",
75      "what_to_find": "section-title",
76      "what_to_do": "getText"
77    },
78    "20161141559027": {
79      "how_to_find": "linkText",
80      "what_to_find": "Log in",
81      "what_to_do": "getCssValue",
82      "type_of_action": "color"
83    },
84    "20161141559028": {
85      "how_to_find": "cssSelector",
86      "what_to_find": "#sb-site > DIV:nth-child(1) > SECTION:nth-child(3) > DIV:
            nth-child(1) > DIV:nth-child(1) > DIV:nth-child(1) > DIV:nth-child(1)
            > DIV:nth-child(1) > DIV:nth-child(1)",
87      "what_to_do": "getText"
88    },
89    "s20161141559019": {
90      "how_to_find": "name",
91      "what_to_find": "user[name]",
92      "what_to_do": "sendKeys"
93    },
94    "s20161141559020": {
95      "how_to_find": "name",
96      "what_to_find": "user[email]",
97      "what_to_do": "sendKeys"
98    },
99    "s20161141559021": {
100     "how_to_find": "name",
101     "what_to_find": "user[password]",
```

```
102     "what_to_do": "sendKeys"
103   },
104   "s20161141559022": {
105     "how_to_find": "name",
106     "what_to_find": "user[password_confirmation]",
107     "what_to_do": "sendKeys"
108   },
109   "20161141559024": {
110     "how_to_find": "cssSelector",
111     "what_to_find": "#new_user > BUTTON:nth-child(7)",
112     "what_to_do": "submit"
113   },
114   "2016114208002": {
115     "how_to_find": "cssSelector",
116     "what_to_find": "#sb-site > DIV:nth-child(1) > SECTION:nth-child(3) > DIV:
          nth-child(1) > DIV:nth-child(1) > DIV:nth-child(1) > DIV:nth-child(1)
          > DIV:nth-child(1) > DIV:nth-child(1)",
117     "what_to_do": "getText"
118   },
119   "2016114208003": {
120     "how_to_find": "cssSelector",
121     "what_to_find": "#sb-site > DIV:nth-child(1) > SECTION:nth-child(3) > DIV:
          nth-child(1) > DIV:nth-child(1) > DIV:nth-child(1) > DIV:nth-child(2)
          > DIV:nth-child(1) > H1:nth-child(1)",
122     "what_to_do": "getText"
123   },
124   "2016114208004": {
125     "how_to_find": "cssSelector",
126     "what_to_find": "#sb-site > DIV:nth-child(1) > SECTION:nth-child(3) > DIV:
          nth-child(1) > DIV:nth-child(1) > DIV:nth-child(1) > DIV:nth-child(3)
          > DIV:nth-child(1) > H1:nth-child(1)",
127     "what_to_do": "getText"
128   },
129   "2016114208007": {
130     "how_to_find": "linkText",
131     "what_to_find": "Welcome admin",
132     "what_to_do": "getText"
133   },
134   "2016114208009": {
135     "how_to_find": "cssSelector",
136     "what_to_find": "#sb-site > DIV:nth-child(1) > SECTION:nth-child(3) > DIV:
          nth-child(1) > DIV:nth-child(1) > DIV:nth-child(1) > DIV:nth-child(2)
          > DIV:nth-child(1) > H1:nth-child(1) > A:nth-child(1) > I:nth-child(1)
          ",
137     "what_to_do": "click"
138   },
139   "20161121029002": {
140     "how_to_find": "linkText",
```

```
141     "what_to_find": "Periodic Table",
142     "what_to_do": "click"
143   },
144   "20161121029018": {
145     "how_to_find": "cssSelector",
146     "what_to_find": "#header-full-top > DIV:nth-child(1) > NAV:nth-child(2) >
            UL:nth-child(2) > LI:nth-child(1) > A:nth-child(1) > I:nth-child(1)",
147     "what_to_do": "click"
148   },
149   "2016114208019": {
150     "how_to_find": "xpath",
151     "what_to_find": "//*[@id='bs-example-navbar-collapse-1']/ul/li[3]",
152     "what_to_do": "getText",
153     "type_of_action": "class"
154   },
155   "2016114208020": {
156     "how_to_find": "name",
157     "what_to_find": "ontology[file]",
158     "what_to_do": "getText"
159   },
160   "2016114208021": {
161     "how_to_find": "className",
162     "what_to_find": "section-title",
163     "what_to_do": "getText"
164   },
165   "s2016114208010": {
166     "how_to_find": "name",
167     "what_to_find": "ontology[name]",
168     "what_to_do": "sendKeys"
169   },
170   "s2016114208011": {
171     "how_to_find": "id",
172     "what_to_find": "ontology_desc",
173     "what_to_do": "sendKeys"
174   },
175   "s2016114208012": {
176     "how_to_find": "cssSelector",
177     "what_to_find": "#new_ontology > DIV:nth-child(5) > LABEL:nth-child(1)",
178     "what_to_do": "click"
179   },
180   "s2016114208014": {
181     "how_to_find": "name",
182     "what_to_find": "ontology[file]",
183     "what_to_do": "sendKeys"
184   },
185   "2016114208016": {
186     "how_to_find": "cssSelector",
187     "what_to_find": "#new_ontology > BUTTON:nth-child(7)",
```

```
188     "what_to_do": "submit"
189   },
190   "20161142080l8": {
191     "how_to_find": "cssSelector",
192     "what_to_find": "#sb-site > DIV:nth-child(1) > SECTION:nth-child(3) > DIV:
          nth-child(1) > DIV:nth-child(1) > DIV:nth-child(1) > DIV:nth-child(1)
          > DIV:nth-child(1) > DIV:nth-child(1)",
193     "what_to_do": "getCssValue",
194     "type_of_action": "background-color"
195   },
196   "20161121029022": {
197     "how_to_find": "cssSelector",
198     "what_to_find": "#sb-site > DIV:nth-child(1) > SECTION:nth-child(3) > DIV:
          nth-child(1) > DIV:nth-child(1) > DIV:nth-child(1) > DIV:nth-child(2)
          > DIV:nth-child(3) > TABLE:nth-child(2) > TBODY:nth-child(2) > TR:nth-
          child(1) > TD:nth-child(2)",
199     "what_to_do": "getText"
200   },
201   "20161121029023": {
202     "how_to_find": "cssSelector",
203     "what_to_find": "#sb-site > DIV:nth-child(1) > SECTION:nth-child(3) > DIV:
          nth-child(1) > DIV:nth-child(1) > DIV:nth-child(1) > DIV:nth-child(2)
          > DIV:nth-child(1) > DL:nth-child(2) > DD:nth-child(8)",
204     "what_to_do": "getText"
205   },
206   "20161121029024": {
207     "how_to_find": "cssSelector",
208     "what_to_find": "#sb-site > DIV:nth-child(1) > SECTION:nth-child(3) > DIV:
          nth-child(1) > DIV:nth-child(1) > DIV:nth-child(1) > DIV:nth-child(2)
          > DIV:nth-child(1) > DL:nth-child(4) > DD:nth-child(6) > A:nth-child
          (1) > BUTTON:nth-child(1)",
209     "what_to_do": "getText"
210   },
211   "20161121029025": {
212     "how_to_find": "cssSelector",
213     "what_to_find": "#sb-site > DIV:nth-child(1) > SECTION:nth-child(3) > DIV:
          nth-child(1) > DIV:nth-child(1) > DIV:nth-child(1) > DIV:nth-child(1)
          > DIV:nth-child(1) > H1:nth-child(1) > A:nth-child(2)",
214     "what_to_do": "getText"
215   },
216   "20161121029026": {
217     "how_to_find": "cssSelector",
218     "what_to_find": "#sb-site > DIV:nth-child(1) > SECTION:nth-child(3) > DIV:
          nth-child(1) > DIV:nth-child(1) > DIV:nth-child(1) > DIV:nth-child(1)
          > DIV:nth-child(3) > DIV:nth-child(1) > DIV:nth-child(1) > H1:nth-
          child(1) > A:nth-child(1)",
219     "what_to_do": "getText"
220   },
```

```
221   "20161121029027": {
222     "how_to_find": "cssSelector",
223     "what_to_find": "#sb-site > DIV:nth-child(1) > SECTION:nth-child(3) > DIV:
          nth-child(1) > DIV:nth-child(1) > DIV:nth-child(1) > DIV:nth-child(1)
          > DIV:nth-child(2) > DIV:nth-child(1) > DIV:nth-child(1) > H1:nth-
          child(1) > A:nth-child(1)",
224     "what_to_do": "getText"
225   },
226   "20161121029028": {
227     "how_to_find": "cssSelector",
228     "what_to_find": "#sb-site > DIV:nth-child(1) > SECTION:nth-child(3) > DIV:
          nth-child(1) > DIV:nth-child(1) > DIV:nth-child(1) > DIV:nth-child(1)
          > DIV:nth-child(1) > DL:nth-child(2) > DD:nth-child(2)",
229     "what_to_do": "getText"
230   },
231   "20161121029029": {
232     "how_to_find": "cssSelector",
233     "what_to_find": "#sb-site > DIV:nth-child(1) > SECTION:nth-child(3) > DIV:
          nth-child(1) > DIV:nth-child(1) > DIV:nth-child(1) > DIV:nth-child(1)
          > DIV:nth-child(1) > DL:nth-child(2) > DD:nth-child(6) > FORM:nth-
          child(1) > BUTTON:nth-child(4)",
234     "what_to_do": "getText"
235   },
236   "20161121029030": {
237     "how_to_find": "cssSelector",
238     "what_to_find": "#sb-site > DIV:nth-child(1) > SECTION:nth-child(3) > DIV:
          nth-child(1) > DIV:nth-child(1) > DIV:nth-child(1) > DIV:nth-child(1)
          > DIV:nth-child(1) > DL:nth-child(4) > DD:nth-child(2)",
239     "what_to_do": "getText"
240   },
241   "20161121029031": {
242     "how_to_find": "cssSelector",
243     "what_to_find": "#sb-site > DIV:nth-child(1) > SECTION:nth-child(3) > DIV:
          nth-child(1) > DIV:nth-child(1) > DIV:nth-child(1) > DIV:nth-child(1)
          > DIV:nth-child(2) > DIV:nth-child(2) > DIV:nth-child(1)",
244     "what_to_do": "getText"
245   },
246   "20161121029016": {
247     "how_to_find": "cssSelector",
248     "what_to_find": "#sb-site > DIV:nth-child(1) > SECTION:nth-child(3) > DIV:
          nth-child(1) > DIV:nth-child(1) > DIV:nth-child(1) > DIV:nth-child(1)
          > DIV:nth-child(1) > H1:nth-child(1) > A:nth-child(2) > I:nth-child(1)
          ",
249     "what_to_do": "click"
250   },
251   "20161121029019": {
252     "how_to_find": "cssSelector",
```

```
253     "what_to_find": "#sb-site > DIV:nth-child(1) > SECTION:nth-child(3) > DIV:
            nth-child(1) > DIV:nth-child(1) > DIV:nth-child(1) > DIV:nth-child(1)
            > DIV:nth-child(2) > DIV:nth-child(1) > DIV:nth-child(1) > H1:nth-
            child(1) > A:nth-child(1)",
254     "what_to_do": "click"
255   },
256   "20161121029021": {
257     "how_to_find": "cssSelector",
258     "what_to_find": "#sb-site > DIV:nth-child(1) > SECTION:nth-child(3) > DIV:
            nth-child(1) > DIV:nth-child(1) > DIV:nth-child(1) > DIV:nth-child(1)
            > DIV:nth-child(3) > DIV:nth-child(1) > DIV:nth-child(1) > H1:nth-
            child(1) > A:nth-child(1) > I:nth-child(1)",
259     "what_to_do": "click"
260   },
261   "2017031127004": {
262     "how_to_find": "id",
263     "what_to_find": "prefixes_add_all",
264     "what_to_do": "getText"
265   },
266   "s20161121029003": {
267     "how_to_find": "id",
268     "what_to_find": "prefixes_select",
269     "what_to_do": "click"
270   },
271   "s20161121029004": {
272     "how_to_find": "id",
273     "what_to_find": "prefixes_add",
274     "what_to_do": "click"
275   },
276   "s20161121029005": {
277     "how_to_find": "id",
278     "what_to_find": "prefixes_add",
279     "what_to_do": "click"
280   },
281   "s20161121029006": {
282     "how_to_find": "id",
283     "what_to_find": "prefixes_add",
284     "what_to_do": "click"
285   },
286   "s20161121029007": {
287     "how_to_find": "name",
288     "what_to_find": "query[timeout]",
289     "what_to_do": "sendKeys"
290   },
291   "s20161121029034": {
292     "how_to_find": "id",
293     "what_to_find": "prefixes_add",
294     "what_to_do": "click"
```

```
295   },
296   "s20161131049005": {
297     "how_to_find": "id",
298     "what_to_find": "prefixes_add",
299     "what_to_do": "click"
300   },
301   "20161121029009": {
302     "how_to_find": "cssSelector",
303     "what_to_find": "#submit_query > DIV:nth-child(5) > BUTTON:nth-child(2)",
304     "what_to_do": "submit"
305   },
306   "20161121425027": {
307     "how_to_find": "cssSelector",
308     "what_to_find": "#sb-site > DIV:nth-child(1) > SECTION:nth-child(3) > DIV:
          nth-child(1) > DIV:nth-child(1) > DIV:nth-child(1) > DIV:nth-child(1)
          > DIV:nth-child(1) > DIV:nth-child(1)",
309     "what_to_do": "getText"
310   },
311   "s20161121425024": {
312     "how_to_find": "cssSelector",
313     "what_to_find": "#sb-site > DIV:nth-child(1) > SECTION:nth-child(3) > DIV:
          nth-child(1) > DIV:nth-child(1) > DIV:nth-child(1) > DIV:nth-child(1)
          > DIV:nth-child(1) > H1:nth-child(1) > A:nth-child(3) > I:nth-child(1)
          ",
314     "what_to_do": "click"
315   },
316   "20161121425023": {
317     "what_to_do": "accept"
318   },
319   "20161121029038": {
320     "how_to_find": "cssSelector",
321     "what_to_find": "#sb-site > DIV:nth-child(1) > SECTION:nth-child(3) > DIV:
          nth-child(1) > DIV:nth-child(1) > DIV:nth-child(1) > DIV:nth-child(1)
          > DIV:nth-child(1) > H1:nth-child(1) > A:nth-child(1) > I:nth-child(1)
          ",
322     "what_to_do": "getText"
323   },
324   "20161121029039": {
325     "how_to_find": "id",
326     "what_to_find": "div_prefixes",
327     "what_to_do": "getText"
328   },
329   "20161121029040": {
330     "how_to_find": "cssSelector",
331     "what_to_find": "#sb-site > DIV:nth-child(1) > SECTION:nth-child(3) > DIV:
          nth-child(1) > DIV:nth-child(1) > DIV:nth-child(1) > DIV:nth-child(4)
          > DIV:nth-child(1)",
332     "what_to_do": "getText"
```

```
333   },
334   "20161121029041": {
335     "how_to_find": "cssSelector",
336     "what_to_find": "#sb-site > DIV:nth-child(1) > SECTION:nth-child(3) > DIV:
          nth-child(1) > DIV:nth-child(1) > DIV:nth-child(1) > DIV:nth-child(4)
          > DIV:nth-child(1) > H1:nth-child(1) > SMALL:nth-child(1)",
337     "what_to_do": "getText"
338   },
339   "20161121029042": {
340     "how_to_find": "className",
341     "what_to_find": "col-md-3",
342     "what_to_do": "getText"
343   },
344   "20161121029043": {
345     "how_to_find": "id",
346     "what_to_find": "query_start_save",
347     "what_to_do": "getText"
348   },
349   "20161121029047": {
350     "how_to_find": "id",
351     "what_to_find": "query_saving",
352     "what_to_do": "getText"
353   },
354   "20161121029012": {
355     "how_to_find": "cssSelector",
356     "what_to_find": "#sb-site > DIV:nth-child(1) > SECTION:nth-child(3) > DIV:
          nth-child(1) > DIV:nth-child(1) > DIV:nth-child(1) > DIV:nth-child(1)
          > DIV:nth-child(1) > H1:nth-child(1) > A:nth-child(1) > I:nth-child(1)
          ",
357     "what_to_do": "click"
358   },
359   "20161121425006": {
360     "how_to_find": "cssSelector",
361     "what_to_find": "#query_save_area > DIV:nth-child(1)",
362     "what_to_do": "getText"
363   },
364   "2017031127005": {
365     "how_to_find": "id",
366     "what_to_find": "query_stop_save",
367     "what_to_do": "getText"
368   },
369   "s20161121425003": {
370     "how_to_find": "id",
371     "what_to_find": "query_start_save",
372     "what_to_do": "click"
373   },
374   "s20161121425004": {
375     "how_to_find": "id",
```

```
376     "what_to_find": "name",
377     "what_to_do": "sendKeys"
378   },
379   "s20161121425005": {
380     "how_to_find": "id",
381     "what_to_find": "desc",
382     "what_to_do": "sendKeys"
383   },
384   "20161121425002": {
385     "how_to_find": "id",
386     "what_to_find": "query_save",
387     "what_to_do": "click"
388   },
389   "20161121425013": {
390     "how_to_find": "linkText",
391     "what_to_find": "Periodic Table",
392     "what_to_do": "getText"
393   },
394   "20161121425014": {
395     "how_to_find": "cssSelector",
396     "what_to_find": "TABLE:nth-child(1) > TBODY:nth-child(2) > TR:nth-child(1)
          > TD:nth-child(3) > A:nth-child(1)",
397     "what_to_do": "getText"
398   },
399   "20161121029014": {
400     "how_to_find": "cssSelector",
401     "what_to_find": "#sb-site > DIV:nth-child(1) > SECTION:nth-child(3) > DIV:
        nth-child(1) > DIV:nth-child(1) > DIV:nth-child(1) > DIV:nth-child(1)
        > DIV:nth-child(1) > H1:nth-child(1) > A:nth-child(1)",
402     "what_to_do": "click"
403   },
404   "20161121425018": {
405     "how_to_find": "cssSelector",
406     "what_to_find": "SECTION:nth-child(3) > DIV:nth-child(1) > DIV:nth-child
        (1) > DIV:nth-child(1) > DIV:nth-child(1) > DIV:nth-child(1) > DIV:nth
        -child(1)",
407     "what_to_do": "getText"
408   },
409   "s20161121425017": {
410     "how_to_find": "cssSelector",
411     "what_to_find": "TABLE:nth-child(1) > TBODY:nth-child(2) > TR:nth-child(1)
          > TD:nth-child(3) > A:nth-child(1)",
412     "what_to_do": "click"
413   },
414   "20161121425016": {
415     "what_to_do": "accept"
416   },
417   "20161121425007": {
```

```
418     "how_to_find": "className",
419     "what_to_find": "section-title",
420     "what_to_do": "getText"
421   },
422   "20161121425008": {
423     "how_to_find": "cssSelector",
424     "what_to_find": "#sb-site > DIV:nth-child(1) > SECTION:nth-child(3) > DIV:
          nth-child(1) > DIV:nth-child(1) > DIV:nth-child(1) > H3:nth-child(2)",
425     "what_to_do": "getText"
426   },
427   "20161121425009": {
428     "how_to_find": "id",
429     "what_to_find": "ontology_desc",
430     "what_to_do": "getText"
431   },
432   "20161121425010": {
433     "how_to_find": "id",
434     "what_to_find": "ontology_public",
435     "what_to_do": "getText"
436   },
437   "20161121425011": {
438     "how_to_find": "cssSelector",
439     "what_to_find": "FORM > BUTTON:nth-child(7)",
440     "what_to_do": "getText"
441   },
442   "20161121425028": {
443     "how_to_find": "name",
444     "what_to_find": "ontology[shared]",
445     "what_to_do": "getText"
446   },
447   "20161121425019": {
448     "how_to_find": "className",
449     "what_to_find": "section-title",
450     "what_to_do": "getText"
451   },
452   "20161121425020": {
453     "how_to_find": "name",
454     "what_to_find": "user[name]",
455     "what_to_do": "getText",
456     "type_of_action": "value"
457   },
458   "20161121425021": {
459     "how_to_find": "cssSelector",
460     "what_to_find": "#edit_user > BUTTON:nth-child(9)",
461     "what_to_do": "getText"
462   },
463   "20161121425025": {
464     "how_to_find": "cssSelector",
```

```
465     "what_to_find": "#sb-site > DIV:nth-child(1) > SECTION:nth-child(3) > DIV:
            nth-child(1) > DIV:nth-child(1) > DIV:nth-child(1) > DIV:nth-child(1)
            > DIV:nth-child(1)",
466     "what_to_do": "getText"
467   },
468   "20161121425026": {
469     "how_to_find": "cssSelector",
470     "what_to_find": "#sb-site > DIV:nth-child(1) > SECTION:nth-child(3) > DIV:
            nth-child(1) > DIV:nth-child(1) > DIV:nth-child(1) > DIV:nth-child(1)
            > DIV:nth-child(1) > H1:nth-child(1) > A:nth-child(1)",
471     "what_to_do": "getText"
472   },
473   "20161121851005": {
474     "how_to_find": "cssSelector",
475     "what_to_find": "#sb-site > DIV:nth-child(1) > SECTION:nth-child(3) > DIV:
            nth-child(1) > DIV:nth-child(1) > DIV:nth-child(1) > DIV:nth-child(1)
            > DIV:nth-child(1) > DIV:nth-child(1)",
476     "what_to_do": "getText"
477   },
478   "20161121851007": {
479     "how_to_find": "cssSelector",
480     "what_to_find": "#sb-site > DIV:nth-child(1) > SECTION:nth-child(3) > DIV:
            nth-child(1) > DIV:nth-child(1) > DIV:nth-child(1) > DIV:nth-child(1)
            > DIV:nth-child(1) > DIV:nth-child(1)",
481     "what_to_do": "getText"
482   },
483   "20161121851008": {
484     "how_to_find": "className",
485     "what_to_find": "lead",
486     "what_to_do": "getText"
487   },
488   "20161131049002": {
489     "how_to_find": "cssSelector",
490     "what_to_find": "#sb-site > DIV:nth-child(1) > SECTION:nth-child(3) > DIV:
            nth-child(1) > DIV:nth-child(1) > DIV:nth-child(1) > DIV:nth-child(1)
            > DIV:nth-child(1) > DIV:nth-child(1)",
491     "what_to_do": "getText"
492   },
493   "20161131049003": {
494     "how_to_find": "className",
495     "what_to_find": "lead",
496     "what_to_do": "getText"
497   },
498   "20161131049006": {
499     "how_to_find": "cssSelector",
500     "what_to_find": "#sb-site > DIV:nth-child(1) > SECTION:nth-child(3) > DIV:
            nth-child(1) > DIV:nth-child(1) > DIV:nth-child(1) > DIV:nth-child(1)
            > DIV:nth-child(1) > DIV:nth-child(1)",
```

```
501     "what_to_do": "getText"
502   },
503   "20161131117002": {
504     "how_to_find": "cssSelector",
505     "what_to_find": "#query_save_area > DIV:nth-child(1)",
506     "what_to_do": "getText"
507   },
508   "20161131117003": {
509     "how_to_find": "cssSelector",
510     "what_to_find": "#query_save_area > DIV:nth-child(1) > UL:nth-child(2) >
          LI:nth-child(1)",
511     "what_to_do": "getText"
512   }
513 }
```

Appendix C

TOM App System Models

C.1 State Machine

```
<?xml version="1.0" encoding="UTF-8"?>
<scxml xmlns="http://www.w3.org/2005/07/scxml" name="TOM-App" initial="0">
   <state id="0">
      <onentry id="2017851526026" type="contains" />
      <onentry id="2017851526028" type="disabled?" />
      <transition id="2017851526010" target="2017851526009" />
      <state id="login" type="form">
         <send label="s2017841236001" type="required" />
         <send label="s2017841236002" type="required" />
         <transition type="form" label="2017841236004">
            <submit target="2017841236005" />
            <error target="2017851622001" />
         </transition>
         <onexit id="2017911432001" type="displayed?" />
      </state>
   </state>
   <state id="2017841236005">
      <onentry id="2017841236028" type="contains" />
      <onentry id="2017841236083" type="contains" />
      <onentry id="2017841236084" type="contains" />
      <onentry id="2017841236085" type="contains" />
      <transition id="2017841236019" target="2017841236018" />
      <transition id="2017841236023" target="2017841236022" />
      <transition id="2017841236047" target="2017841236046" />
      <transition id="2017841236079" target="0" />
      <state id="Json" type="form">
         <send label="s2017841236067" type="required" />
         <send label="s2017841236068" type="required" />
         <send label="s2017851526034" type="required" element="checkbox" />
         <send label="s2017851526035" type="required" element="checkbox" />
         <transition type="form" label="2017841236064">
            <submit target="2017841236008" />
         </transition>
```

```
34          </state>
35          <state id="IRIT" type="form">
36             <send label="s2017911626010" type="required" element="checkbox" />
37             <send label="s2017911626011" type="required" element="checkbox" />
38             <send label="s2017911626012" type="required" />
39             <send label="s2017911626013" type="required" />
40             <transition type="form" label="2017911626008">
41                <submit target="2017841236008" />
42                <error target="2017911626009" />
43             </transition>
44          </state>
45          <state id="WEB" type="form">
46             <send label="s2017911626026" type="required" element="checkbox" />
47             <send label="s2017911626027" type="required" element="checkbox" />
48             <send label="s2017911626028" type="required" />
49             <send label="s2017911626029" type="required" element="checkbox" />
50             <send label="s2017911626030" type="required" />
51             <send label="s2017911626032" type="required" />
52             <transition type="form" label="2017911626024">
53                <submit target="2017841236008" />
54                <error target="2017911626025" />
55             </transition>
56          </state>
57       </state>
58       <state id="2017841236008">
59          <onentry id="2017841236033" type="contains" />
60          <onentry id="2017841236130" type="enabled?" />
61          <onentry id="2017911626020" type="not_displayed?" />
62          <transition id="2017851526003" target="2017851526002" />
63          <state id="Choose" type="form">
64             <send label="s2017841236072" type="required" element="checkbox" />
65             <transition type="form" label="2017841236070">
66                <submit target="2017841236010" />
67                <error target="2017911626017" />
68             </transition>
69             <onexit id="2017921517002" type="contains" />
70          </state>
71       </state>
72       <state id="2017841236010">
73          <onentry id="2017841236037" type="contains" />
74          <onentry id="2017841236131" type="displayed?" />
75          <transition id="2017841236013" target="2017841236012" />
76          <state id="choose" type="form">
77             <send label="s2017841236075" type="optional" element="checkbox" />
78             <send label="s2017841236076" type="optional" element="checkbox" />
79             <transition type="form" label="2017841236074">
80                <submit target="2017841236012" />
81             </transition>
```

```
82          </state>
83       </state>
84       <state id="2017841236012">
85          <onentry id="2017841236041" type="contains" />
86          <onentry id="2017841236132" type="displayed?" />
87          <onentry id="2017841236133" type="displayed?" />
88          <onentry id="2017841236134" type="displayed?" />
89          <transition id="2017841236016" target="2017841236015" />
90       </state>
91       <state id="2017841236015">
92          <onentry id="2017911626016" type="contains" />
93          <transition id="2017841236017" target="2017841236005" />
94          <transition id="2017811930002" target="2017841236012" />
95       </state>
96       <state id="2017841236018">
97          <onentry id="2017841236118" type="contains" />
98          <onentry id="2017841236119" type="contains" />
99          <onentry id="2017841236135" type="not_displayed?" />
100         <transition id="2017841236021" target="2017841236020" />
101      </state>
102      <state id="2017841236020">
103         <onentry id="2017841236120" type="contains" />
104         <onentry id="2017841236121" type="displayed?" />
105         <onentry id="2017841236136" type="displayed?" />
106      </state>
107      <state id="2017841236022">
108         <onentry id="2017841236122" type="contains" />
109         <onentry id="2017841236123" type="contains" />
110         <onentry id="2017841236137" type="not_displayed?" />
111         <transition id="2017841236025" target="2017841236024" />
112         <transition id="2017841236089" target="2017841236087" />
113      </state>
114      <state id="2017841236024">
115         <onentry id="2017841236124" type="contains" />
116         <onentry id="2017841236125" type="enabled?" />
117         <onentry id="2017841236126" type="enabled?" />
118         <onentry id="2017841236138" type="displayed?" />
119      </state>
120      <state id="2017841236046">
121         <onentry id="2017841236048" type="contains" />
122         <onentry id="2017841236128" type="displayed?" />
123         <transition id="2017811930001" target="2017841236018" />
124      </state>
125      <state id="2017841236087">
126         <onentry id="2017841236140" type="contains" />
127         <state id="add" type="form">
128            <send label="s2017841236143" type="required" />
129            <send label="s2017851526001" type="required" />
```

```
130         <transition type="form" label="2017841236142">
131            <submit target="2017841236022" />
132            <error target="2017851622011" />
133         </transition>
134      </state>
135   </state>
136   <state id="2017851526002">
137      <onentry id="2017851526004" type="contains" />
138      <state id="Add" type="form">
139         <send label="s2017851526007" type="required" />
140         <send label="s2017851526008" type="required" />
141         <transition type="form" label="2017851526006">
142            <submit target="2017841236008" />
143            <error target="2017851622009" />
144         </transition>
145      </state>
146   </state>
147   <state id="2017851526009">
148      <onentry id="2017851526023" type="disabled?" />
149      <onentry id="2017851526024" type="contains" />
150      <transition id="2017851526022" target="0" />
151      <state id="" type="form">
152         <send label="s2017851526011" type="required" />
153         <send label="s2017851526012" type="required" />
154         <send label="s2017851526013" type="required" />
155         <transition type="form" label="2017851526015">
156            <submit target="201791958001" />
157            <error target="201791958002" />
158         </transition>
159         <onexit id="2017911432006" type="displayed?" />
160      </state>
161   </state>
162   <state id="2017851622001">
163      <onentry id="2017851622004" type="displayed?" />
164      <onentry id="201791105009" type="contains" />
165   </state>
166   <state id="2017851622009">
167      <onentry id="2017851622010" type="not_displayed?" />
168      <onentry id="201791105003" type="contains" />
169   </state>
170   <state id="2017851622011">
171      <onentry id="2017851622012" type="not_displayed?" />
172      <onentry id="201791105004" type="contains" />
173   </state>
174   <state id="201791958001">
175      <onentry id="201791105005" type="contains" />
176      <onentry id="201791105006" type="contains" />
177   </state>
```

```
178    <state id="201791958002">
179       <onentry id="201791105007" type="displayed?" />
180       <onentry id="201791105008" type="contains" />
181    </state>
182    <state id="2017911626009">
183       <onentry id="2017911626014" type="contains" />
184       <onentry id="2017911626022" type="not_displayed?" />
185    </state>
186    <state id="2017911626025">
187       <onentry id="2017911626014" type="contains" />
188       <onentry id="2017911626022" type="not_displayed?" />
189    </state>
190    <state id="2017911626017">
191       <onentry id="2017911626018" type="contains" />
192       <onentry id="2017911626019" type="displayed?" />
193       <onentry id="2017911626021" type="not_displayed?" />
194    </state>
195 </scxml>
```

C.2 Values File

```
1  [
2    { "2017851526026" : "Log in with your account" },
3    { "s2017841236001" : "tester@tester.com" },
4    { "s2017841236002" : "tester" },
5    { "2017841236028" : "Generation" },
6    { "2017841236083" : "Type of tests to generate:" },
7    { "2017841236084" : "Set up the algorithm:" },
8    { "2017841236085" : "Algorithm configuration:" },
9    { "s2017841236067" : "1" },
10   { "s2017841236068" : "1" },
11   { "s2017851526034" : "" },
12   { "s2017851526035" : "" },
13   { "s2017911626010" : "" },
14   { "s2017911626011" : "" },
15   { "s2017911626012" : "1" },
16   { "s2017911626013" : "2" },
17   { "s2017911626026" : "" },
18   { "s2017911626027" : "" },
19   { "s2017911626028" : "www.google.pt" },
20   { "s2017911626029" : "" },
21   { "s2017911626030" : "1" },
```

```
22   { "s2017911626032" : "2" },
23   { "2017841236033" : "Choose the project to use" },
24   { "2017921517002" : "Mutations" },
25   { "s2017841236072" : "" },
26   { "2017841236037" : "Mutations" },
27   { "s2017841236075" : "" },
28   { "s2017841236076" : "" },
29   { "2017841236041" : "Generation Request Preview" },
30   { "2017911626016" : "Do you want to continue" },
31   { "2017841236118" : "Generation Results" },
32   { "2017841236119" : "Generation ID:" },
33   { "2017841236120" : "Configuration:" },
34   { "2017841236122" : "Project management" },
35   { "2017841236123" : "Name:" },
36   { "2017841236124" : "Project content:" },
37   { "2017841236048" : "Notifications" },
38   { "2017841236140" : "Upload a new Project" },
39   { "s2017841236143" : "Project" },
40   { "s2017851526001" : "/Users/mgonc/Desktop/Editor/state_machine.xml" },
41   { "2017851526004" : "Upload a new Project" },
42   { "s2017851526007" : "Project" },
43   { "s2017851526008" : "/Users/mgonc/Desktop/Editor/state_machine.xml" },
44   { "2017851526024" : "Register your account" },
45   { "s2017851526011" : "Marcelo" },
46   { "s2017851526012" : "marcelo.rgonc@gmail.com" },
47   { "s2017851526013" : "1234567" },
48   { "201791105009" : "Log in with your account" },
49   { "201791105003" : "Upload a new Project" },
50   { "201791105004" : "Upload a new Project" },
51   { "201791105005" : "Generation" },
52   { "201791105006" : "Marcelo" },
53   { "201791105008" : "Register your account" },
54   { "2017911626014" : "Generation" },
55   { "2017911626014" : "Generation" },
56   { "2017911626018" : "Choose the project to use" }
57 ]
```

C.3 Mutations File


```
[
   {
      "type":"mistake",
      "model_element":"s2017841236001",
      "value":"tester2@tester.com",
      "fail":"1"
   },
   {
      "type":"mistake",
      "model_element":"s2017841236002",
      "value":"1234567",
      "fail":"1"
   },
   {
      "type":"mistake",
      "model_element":"s2017911626012",
      "value":"6",
      "fail":"1"
   },
   {
      "type":"lapse",
      "model_element":"s2017911626028",
      "fail":"1"
   },
   {
      "type":"lapse",
      "model_element":"s2017911626029",
      "fail":"1"
   },
   {
      "type":"lapse",
      "model_element":"s2017841236072",
      "fail":"1"
   },
   {
      "type":"slip",
      "model_element":"s2017841236075",
      "fail":"0"
   },
   {
      "type":"lapse",
      "model_element":"s2017841236143",
      "fail":"1"
   },
   {
```

```
46        "type":"lapse",
47        "model_element":"s2017851526007",
48        "fail":"1"
49     },
50     {
51        "type":"lapse",
52        "model_element":"s2017851526008",
53        "fail":"1"
54     },
55     {
56        "type":"mistake",
57        "model_element":"s2017851526012",
58        "value":"tester@tester.com",
59        "fail":"1"
60     }
61 ]
```

C.4 Mapping File

```
1  {
2     "2017851526026":{
3        "how_to_find":"className",
4        "what_to_find":"form-signin-heading",
5        "what_to_do":"getText"
6     },
7     "2017851526028":{
8        "how_to_find":"cssSelector",
9        "what_to_find":"HTML > BODY:nth-child(2) > APP-ROOT:nth-child(1) > APP-
            LOGIN:nth-child(2) > DIV:nth-child(1) > FORM:nth-child(1) > DIV:nth-
            child(4) > BUTTON:nth-child(1)",
10       "what_to_do":"getText"
11    },
12    "2017851526010":{
13       "how_to_find":"cssSelector",
14       "what_to_find":"HTML > BODY:nth-child(2) > APP-ROOT:nth-child(1) > APP-
            LOGIN:nth-child(2) > DIV:nth-child(1) > FORM:nth-child(1) > DIV:nth-
            child(4) > BUTTON:nth-child(2)",
15       "what_to_do":"click"
16    },
17    "2017911432001":{
18       "how_to_find":"linkText",
19       "what_to_find":"TOM",
```

```
20        "what_to_do":"getText"
21    },
22    "s2017841236001":{
23        "how_to_find":"id",
24        "what_to_find":"email",
25        "what_to_do":"sendKeys"
26    },
27    "s2017841236002":{
28        "how_to_find":"id",
29        "what_to_find":"password",
30        "what_to_do":"sendKeys"
31    },
32    "2017841236004":{
33        "how_to_find":"cssSelector",
34        "what_to_find":"HTML > BODY:nth-child(2) > APP-ROOT:nth-child(1) > APP-
              LOGIN:nth-child(2) > DIV:nth-child(1) > FORM:nth-child(1) > DIV:nth-
              child(4) > BUTTON:nth-child(1)",
35        "what_to_do":"submit"
36    },
37    "2017841236028":{
38        "how_to_find":"className",
39        "what_to_find":"title",
40        "what_to_do":"getText"
41    },
42    "2017841236083":{
43        "how_to_find":"cssSelector",
44        "what_to_find":"#page-content-wrapper > DIV:nth-child(1) > APP-
              GENERATION:nth-child(2) > SECTION:nth-child(1) > DIV:nth-child(2) >
              DIV:nth-child(1) > DIV:nth-child(1) > DIV:nth-child(1)",
45        "what_to_do":"getText"
46    },
47    "2017841236084":{
48        "how_to_find":"cssSelector",
49        "what_to_find":"#page-content-wrapper > DIV:nth-child(1) > APP-
              GENERATION:nth-child(2) > SECTION:nth-child(1) > DIV:nth-child(2) >
              DIV:nth-child(2) > DIV:nth-child(1) > DIV:nth-child(1)",
50        "what_to_do":"getText"
51    },
52    "2017841236085":{
53        "how_to_find":"cssSelector",
54        "what_to_find":"#page-content-wrapper > DIV:nth-child(1) > APP-
              GENERATION:nth-child(2) > SECTION:nth-child(1) > DIV:nth-child(2) >
              DIV:nth-child(2) > DIV:nth-child(2) > DIV:nth-child(1)",
55        "what_to_do":"getText"
56    },
57    "2017841236019":{
58        "how_to_find":"linkText",
59        "what_to_find":"Generation Results",
```

```
60          "what_to_do":"click"
61      },
62      "2017841236023":{
63          "how_to_find":"linkText",
64          "what_to_find":"Projects",
65          "what_to_do":"click"
66      },
67      "2017841236047":{
68          "how_to_find":"cssSelector",
69          "what_to_find":"#notification-link > SPAN:nth-child(1)",
70          "what_to_do":"click"
71      },
72      "2017841236079":{
73          "how_to_find":"cssSelector",
74          "what_to_find":"HTML > BODY:nth-child(2) > APP-ROOT:nth-child(1) > APP-
                HOME:nth-child(2) > NAV:nth-child(1) > DIV:nth-child(2) > UL:nth-
                child(1) > LI:nth-child(3) > SPAN:nth-child(1) > A:nth-child(1) >
                SPAN:nth-child(1)",
75          "what_to_do":"click"
76      },
77      "s2017841236067":{
78          "how_to_find":"name",
79          "what_to_find":"vertex",
80          "what_to_do":"sendKeys"
81      },
82      "s2017841236068":{
83          "how_to_find":"name",
84          "what_to_find":"edge",
85          "what_to_do":"sendKeys"
86      },
87      "s2017851526034":{
88          "how_to_find":"id",
89          "what_to_find":"JSON",
90          "what_to_do":"click"
91      },
92      "s2017851526035":{
93          "how_to_find":"id",
94          "what_to_find":"BFS",
95          "what_to_do":"click"
96      },
97      "2017841236064":{
98          "how_to_find":"id",
99          "what_to_find":"next",
100         "what_to_do":"click"
101     },
102     "s2017911626010":{
103         "how_to_find":"id",
104         "what_to_find":"IRIT",
```

```
105         "what_to_do":"click"
106      },
107      "s2017911626011":{
108         "how_to_find":"id",
109         "what_to_find":"BFS",
110         "what_to_do":"click"
111      },
112      "s2017911626012":{
113         "how_to_find":"name",
114         "what_to_find":"vertex",
115         "what_to_do":"sendKeys"
116      },
117      "s2017911626013":{
118         "how_to_find":"name",
119         "what_to_find":"edge",
120         "what_to_do":"sendKeys"
121      },
122      "2017911626008":{
123         "how_to_find":"id",
124         "what_to_find":"next",
125         "what_to_do":"click"
126      },
127      "s2017911626026":{
128         "how_to_find":"id",
129         "what_to_find":"WEB",
130         "what_to_do":"click"
131      },
132      "s2017911626027":{
133         "how_to_find":"id",
134         "what_to_find":"DFS",
135         "what_to_do":"click"
136      },
137      "s2017911626028":{
138         "how_to_find":"id",
139         "what_to_find":"url",
140         "what_to_do":"sendKeys"
141      },
142      "s2017911626029":{
143         "how_to_find":"id",
144         "what_to_find":"1",
145         "what_to_do":"click"
146      },
147      "s2017911626030":{
148         "how_to_find":"name",
149         "what_to_find":"vertex",
150         "what_to_do":"sendKeys"
151      },
152      "s2017911626032":{
```

```
153       "how_to_find":"name",
154       "what_to_find":"edge",
155       "what_to_do":"sendKeys"
156    },
157    "2017911626024":{
158       "how_to_find":"id",
159       "what_to_find":"next",
160       "what_to_do":"click"
161    },
162    "2017841236033":{
163       "how_to_find":"className",
164       "what_to_find":"title",
165       "what_to_do":"getText"
166    },
167    "2017841236130":{
168       "how_to_find":"linkText",
169       "what_to_find":"Previous",
170       "what_to_do":"getText"
171    },
172    "2017911626020":{
173       "how_to_find":"id",
174       "what_to_find":"next",
175       "what_to_do":"getText"
176    },
177    "2017851526003":{
178       "how_to_find":"id",
179       "what_to_find":"new",
180       "what_to_do":"click"
181    },
182    "2017921517002":{
183       "how_to_find":"className",
184       "what_to_find":"title",
185       "what_to_do":"getText"
186    },
187    "s2017841236072":{
188       "how_to_find":"cssSelector",
189       "what_to_find":"#accordion-group > DIV:nth-child(1) > DIV:nth-child(1)",
190       "what_to_do":"click"
191    },
192    "2017841236070":{
193       "how_to_find":"id",
194       "what_to_find":"next",
195       "what_to_do":"click"
196    },
197    "2017841236037":{
198       "how_to_find":"className",
199       "what_to_find":"title",
200       "what_to_do":"getText"
```

```
201    },
202    "2017841236131":{
203       "how_to_find":"id",
204       "what_to_find":"next",
205       "what_to_do":"getText"
206    },
207    "2017841236013":{
208       "how_to_find":"linkText",
209       "what_to_find":"Next",
210       "what_to_do":"click"
211    },
212    "s2017841236075":{
213       "how_to_find":"id",
214       "what_to_find":"LAPSE_FORM",
215       "what_to_do":"click"
216    },
217    "s2017841236076":{
218       "how_to_find":"id",
219       "what_to_find":"SLIP_CALL",
220       "what_to_do":"click"
221    },
222    "2017841236074":{
223       "how_to_find":"id",
224       "what_to_find":"next",
225       "what_to_do":"click"
226    },
227    "2017841236041":{
228       "how_to_find":"className",
229       "what_to_find":"title",
230       "what_to_do":"getText"
231    },
232    "2017841236132":{
233       "how_to_find":"id",
234       "what_to_find":"run",
235       "what_to_do":"getText"
236    },
237    "2017841236133":{
238       "how_to_find":"cssSelector",
239       "what_to_find":"#page-content-wrapper > DIV:nth-child(1) > APP-PREVIEW:
              nth-child(2) > SECTION:nth-child(1) > DIV:nth-child(2) > DIV:nth-
              child(1)",
240       "what_to_do":"getText"
241    },
242    "2017841236134":{
243       "how_to_find":"cssSelector",
244       "what_to_find":"#page-content-wrapper > DIV:nth-child(1) > APP-PREVIEW:
              nth-child(2) > SECTION:nth-child(1) > DIV:nth-child(3) > DIV:nth-
              child(1) > DIV:nth-child(1)",
```

```
245         "what_to_do":"getText"
246     },
247     "2017841236016":{
248         "how_to_find":"id",
249         "what_to_find":"run",
250         "what_to_do":"click"
251     },
252     "2017911626016":{
253         "how_to_find":"id",
254         "what_to_find":"modal-title",
255         "what_to_do":"getText"
256     },
257     "2017841236017":{
258         "how_to_find":"id",
259         "what_to_find":"confirm",
260         "what_to_do":"click"
261     },
262     "2017811930002":{
263         "how_to_find":"id",
264         "what_to_find":"cancel",
265         "what_to_do":"click"
266     },
267     "2017841236118":{
268         "how_to_find":"cssSelector",
269         "what_to_find":"#page-content-wrapper > DIV:nth-child(1) > APP-HISTORY:
                nth-child(2) > SECTION:nth-child(1) > DIV:nth-child(1) > DIV:nth-
                child(1)",
270         "what_to_do":"getText"
271     },
272     "2017841236119":{
273         "how_to_find":"cssSelector",
274         "what_to_find":"#accordion-group > DIV:nth-child(1) > DIV:nth-child(1) >
                 DIV:nth-child(1) > DIV:nth-child(1) > DIV:nth-child(1) > SPAN:nth-
                child(1) > B:nth-child(1)",
275         "what_to_do":"click"
276     },
277     "2017841236135":{
278         "how_to_find":"cssSelector",
279         "what_to_find":"#accordion-group > DIV:nth-child(1) > DIV:nth-child(2) >
                 DIV:nth-child(1)",
280         "what_to_do":"getText"
281     },
282     "2017841236021":{
283         "how_to_find":"cssSelector",
284         "what_to_find":"#accordion-group > DIV:nth-child(1) > DIV:nth-child(1)",
285         "what_to_do":"click"
286     },
287     "2017841236120":{
```

```
288         "how_to_find":"cssSelector",
289         "what_to_find":"#body > P:nth-child(1)",
290         "what_to_do":"getText"
291     },
292     "2017841236121":{
293         "how_to_find":"cssSelector",
294         "what_to_find":"#body > DIV:nth-child(7) > BUTTON:nth-child(1) > LABEL:
                nth-child(2)",
295         "what_to_do":"getText",
296         "type_of_action":"Delete"
297     },
298     "2017841236136":{
299         "how_to_find":"cssSelector",
300         "what_to_find":"#accordion-group > DIV:nth-child(1) > DIV:nth-child(2) >
                 DIV:nth-child(1)",
301         "what_to_do":"getText"
302     },
303     "2017841236122":{
304         "how_to_find":"cssSelector",
305         "what_to_find":"#page-content-wrapper > DIV:nth-child(1) > APP-FILES:nth
                -child(2) > SECTION:nth-child(1) > DIV:nth-child(1) > DIV:nth-child
                (1)",
306         "what_to_do":"getText"
307     },
308     "2017841236123":{
309         "how_to_find":"cssSelector",
310         "what_to_find":"#accordion-group > DIV:nth-child(1) > DIV:nth-child(1) >
                 DIV:nth-child(1) > DIV:nth-child(1) > DIV:nth-child(1) > SPAN:nth-
                child(1) > B:nth-child(1)",
311         "what_to_do":"getText"
312     },
313     "2017841236137":{
314         "how_to_find":"cssSelector",
315         "what_to_find":"#accordion-group > DIV:nth-child(1) > DIV:nth-child(2) >
                 DIV:nth-child(1)",
316         "what_to_do":"getText"
317     },
318     "2017841236025":{
319         "how_to_find":"cssSelector",
320         "what_to_find":"#accordion-group > DIV:nth-child(1) > DIV:nth-child(1)",
321         "what_to_do":"click"
322     },
323     "2017841236089":{
324         "how_to_find":"id",
325         "what_to_find":"new",
326         "what_to_do":"click"
327     },
328     "2017841236124":{
```

```
329       "how_to_find":"cssSelector",
330       "what_to_find":"#body > P:nth-child(1)",
331       "what_to_do":"getText"
332    },
333    "2017841236125":{
334       "how_to_find":"cssSelector",
335       "what_to_find":"#body > DIV:nth-child(3) > BUTTON:nth-child(1) > SPAN:
              nth-child(1) > I:nth-child(1)",
336       "what_to_do":"getText"
337    },
338    "2017841236126":{
339       "how_to_find":"cssSelector",
340       "what_to_find":"#body > DIV:nth-child(3) > BUTTON:nth-child(2) > SPAN:
              nth-child(2) > I:nth-child(1)",
341       "what_to_do":"getText"
342    },
343    "2017841236138":{
344       "how_to_find":"cssSelector",
345       "what_to_find":"#accordion-group > DIV:nth-child(1) > DIV:nth-child(2) >
               DIV:nth-child(1)",
346       "what_to_do":"getText"
347    },
348    "2017841236048":{
349       "how_to_find":"id",
350       "what_to_find":"notification-title",
351       "what_to_do":"getText"
352    },
353    "2017841236128":{
354       "how_to_find":"id",
355       "what_to_find":"notification-container",
356       "what_to_do":"getText"
357    },
358    "2017811930001":{
359       "how_to_find":"cssSelector",
360       "what_to_find":"#notifications-body > DIV:nth-child(1) > DIV:nth-child
              (1) > A:nth-child(1) > SPAN:nth-child(2)",
361       "what_to_do":"click"
362    },
363    "2017841236140":{
364       "how_to_find":"id",
365       "what_to_find":"modal-title",
366       "what_to_do":"getText"
367    },
368    "s2017841236143":{
369       "how_to_find":"id",
370       "what_to_find":"name",
371       "what_to_do":"sendKeys"
372    },
```

```
373    "s2017851526001":{
374       "how_to_find":"id",
375       "what_to_find":"model-input",
376       "what_to_do":"sendKeys"
377    },
378    "2017841236142":{
379       "how_to_find":"id",
380       "what_to_find":"submit",
381       "what_to_do":"click"
382    },
383    "2017851526004":{
384       "how_to_find":"id",
385       "what_to_find":"modal-title",
386       "what_to_do":"getText"
387    },
388    "s2017851526007":{
389       "how_to_find":"id",
390       "what_to_find":"name",
391       "what_to_do":"sendKeys"
392    },
393    "s2017851526008":{
394       "how_to_find":"id",
395       "what_to_find":"model-input",
396       "what_to_do":"sendKeys"
397    },
398    "2017851526006":{
399       "how_to_find":"id",
400       "what_to_find":"submit",
401       "what_to_do":"click"
402    },
403    "2017851526023":{
404       "how_to_find":"cssSelector",
405       "what_to_find":"HTML > BODY:nth-child(2) > APP-ROOT:nth-child(1) > APP-
              REGISTER:nth-child(2) > DIV:nth-child(1) > FORM:nth-child(1) > DIV:
              nth-child(5) > BUTTON:nth-child(1)",
406       "what_to_do":"getText"
407    },
408    "2017851526024":{
409       "how_to_find":"className",
410       "what_to_find":"form-signin-heading",
411       "what_to_do":"getText"
412    },
413    "2017851526022":{
414       "how_to_find":"cssSelector",
415       "what_to_find":"HTML > BODY:nth-child(2) > APP-ROOT:nth-child(1) > APP-
              REGISTER:nth-child(2) > DIV:nth-child(1) > FORM:nth-child(1) > DIV:
              nth-child(5) > BUTTON:nth-child(2)",
416       "what_to_do":"click"
```

```
417    },
418    "2017911432006":{
419       "how_to_find":"linkText",
420       "what_to_find":"TOM",
421       "what_to_do":"getText"
422    },
423    "s2017851526011":{
424       "how_to_find":"id",
425       "what_to_find":"name",
426       "what_to_do":"sendKeys"
427    },
428    "s2017851526012":{
429       "how_to_find":"id",
430       "what_to_find":"email",
431       "what_to_do":"sendKeys"
432    },
433    "s2017851526013":{
434       "how_to_find":"id",
435       "what_to_find":"password",
436       "what_to_do":"sendKeys"
437    },
438    "2017851526015":{
439       "how_to_find":"id",
440       "what_to_find":"register",
441       "what_to_do":"submit"
442    },
443    "2017851622004":{
444       "how_to_find":"id",
445       "what_to_find":"error",
446       "what_to_do":"getText"
447    },
448    "201791105009":{
449       "how_to_find":"className",
450       "what_to_find":"form-signin-heading",
451       "what_to_do":"getText"
452    },
453    "2017851622010":{
454       "how_to_find":"id",
455       "what_to_find":"submit",
456       "what_to_do":"getText"
457    },
458    "201791105003":{
459       "how_to_find":"className",
460       "what_to_find":"modal-title pull-left",
461       "what_to_do":"getText"
462    },
463    "2017851622012":{
464       "how_to_find":"id",
```

```
465         "what_to_find":"submit",
466         "what_to_do":"getText"
467     },
468     "201791105004":{
469         "how_to_find":"className",
470         "what_to_find":"modal-title pull-left",
471         "what_to_do":"getText"
472     },
473     "201791105005":{
474         "how_to_find":"className",
475         "what_to_find":"title",
476         "what_to_do":"getText"
477     },
478     "201791105006":{
479         "how_to_find":"cssSelector",
480         "what_to_find":"HTML > BODY:nth-child(2) > APP-ROOT:nth-child(1) > APP-
                HOME:nth-child(2) > NAV:nth-child(1) > DIV:nth-child(2) > UL:nth-
                child(1) > LI:nth-child(1) > SPAN:nth-child(1)",
481         "what_to_do":"getText"
482     },
483     "201791105007":{
484         "how_to_find":"cssSelector",
485         "what_to_find":"HTML > BODY:nth-child(2) > APP-ROOT:nth-child(1) > APP-
                REGISTER:nth-child(2) > DIV:nth-child(1) > FORM:nth-child(1) > DIV:
                nth-child(5) > DIV:nth-child(3)",
486         "what_to_do":"getText"
487     },
488     "201791105008":{
489         "how_to_find":"className",
490         "what_to_find":"form-signin-heading",
491         "what_to_do":"getText"
492     },
493     "2017911626014":{
494         "how_to_find":"className",
495         "what_to_find":"title",
496         "what_to_do":"getText"
497     },
498     "2017911626022":{
499         "how_to_find":"id",
500         "what_to_find":"next",
501         "what_to_do":"getText"
502     },
503     "2017911626018":{
504         "how_to_find":"className",
505         "what_to_find":"tittle",
506         "what_to_do":"getText"
507     },
508     "2017911626019":{
```

```
509         "how_to_find":"cssSelector",
510         "what_to_find":"#page-content-wrapper > DIV:nth-child(1) > APP-CHOOSE-
                PROJECT:nth-child(2) > SECTION:nth-child(1) > DIV:nth-child(1) > DIV
                :nth-child(2) > SPAN:nth-child(1) > A:nth-child(2) > I:nth-child(1)"
                ,
511         "what_to_do":"getText"
512     },
513     "2017911626021":{
514         "how_to_find":"id",
515         "what_to_find":"next",
516         "what_to_do":"getText"
517     }
518 }
```